SECURITY VERIFICATION OF SECURE MANET ROUTING PROTOCOLS

THESIS

Matthew F. Steele, Captain, USAF

AFIT/GCS/ENG/12-03

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED

The views expressed in this thesis are those of the author and do not reflect the official policy or position of the United States Air Force, the Department of Defense, or the United States Government.

AFIT/GCS/ENG/12-03

SECURITY VERIFICATION OF SECURE MANET ROUTING PROTOCOLS

THESIS

Presented to the Faculty

Department of Electrical and Computer Engineering

Graduate School of Engineering and Management

Air Force Insitute of Technology

Air University

Air Education and Training Command

in Partial Fulfillment of the Requirements for the

Degree of Master of Science

Matthew F. Steele, B.S.E.E.

Captain, USAF

March 2012

AFIT/GCS/ENG/12-03

SECURITY VERIFICATION OF SECURE MANET ROUTING PROTOCOLS

Matthew F. Steele, B.S.E.E.
Captain, USAF

Approved:

Maj Todd R. Andel, PhD (Chairman)   29 FEB 12
Date

Lt Col Jeffrey W. Humphries, PhD (Member)   29 Feb 12
Date

Maj Jeffrey M. Hemmes, PhD (Member)   29 Feb 12
Date

Abstract

Secure mobile ad hoc network (MANET) routing protocols are not tested thoroughly against their security properties. Previous research focuses on verifying secure, reactive, accumulation-based routing protocols. An improved methodology and framework for secure MANET routing protocol verification is proposed which includes table-based and proactive protocols.

The model checker, SPIN, is selected as the core of the secure MANET verification framework. Security is defined by both accuracy and availability: *a protocol forms accurate routes and these routes are always accurate*. The framework enables exhaustive verification of protocols and results in a counter-example if the protocol is deemed insecure.

The framework is applied to models of the Optimized Link-State Routing (OLSR) and Secure OLSR protocol against five attack vectors. These vectors are based on known attacks against each protocol. Vulnerabilities consistent with published findings are automatically revealed. No unknown attacks were found; however, future attack vectors may lead to new attacks.

The new framework for verifying secure MANET protocols extends verification capabilities to table-based and proactive protocols. More work is needed to create attack vectors that reveal unknown attacks against secure protocols, but the framework makes construction of such vectors easy.

I dedicate this thesis to my loving family. Their support has provided me with the encouragement and environment needed to be successful throughout my time at AFIT.

Acknowledgments

I am indebted to my fellow citizens and the United States Air Force who have made it possible for me to attend the Air Force Institute of Technology.

This thesis was only possible through the advice and mentorship of my advisor Todd Andel.

Matthew F. Steele

Table of Contents

List of Figures

List of Tables

SECURITY VERIFICATION OF SECURE
MANET ROUTING PROTOCOLS

1 Introduction

It may never be possible to prove that specific properties of routing protocols are true, but formal verification methods can demonstrate when properties are false. Routing protocols can be designed and suited for almost any environment from planning logistics routes to routing packetized information over several wireless radio links. Mobile ad hoc networks (MANETs) are a specific example of wireless routing protocols. Recently, many secure protocols have been proposed with the intended property of route security. A formal verification framework for security properties of MANET routing protocols can provide a method to form concise protocol, property, and environmental definitions; to study a protocol within a distributed system domain; to execute the protocol in an execution environment that is conceptually near a physical implementation; and to improve discourse over protocol properties.

1.1 MANETs and Security

The point-to-multipoint architecture, commonly referred to as access point or base station mode, for wireless technology is well established; however, such architecture is infeasible in military, disaster response, and remote sensor applications. This infeasibility is apparent because point-to-multipoint architectures are dependent on infrastructure that is not available in these applications. Forward deployed military units must provide their own methods for communication. In disaster response scenarios communication systems are often destroyed or overloaded, requiring rescue teams to provide their own communication infrastructure. In remote sensor applications wireless sensors are deployed in inhospitable environments with limited battery power and random sensor placement.

An alternative to the point-to-multipoint architecture is ad hoc architecture, referred to as a MANET. MANET protocols are designed to network many wireless nodes independently of any centralized hardware. Every node is responsible for successfully routing information between other nodes. Effectively, all nodes are routers. These protocols adapt to changes in radio link status, making them ideal for wireless communication, especially in scenarios where nodes are constantly moving and no centralized hardware can coordinate communication. The flexibility of MANET protocols also makes them vulnerable to attacks.

Routes consist of a series of links, also known as *hops*, between neighboring nodes. The goal of a MANET is to provide accurate routes between all nodes. An adversary can affect route availability by overt jamming, preventing route formation, or injecting inaccurate routes. A route is inaccurate when it contains one or more non-existent links. An adversary might deploy malicious nodes with the intent of adding inaccurate routes or blocking accurate ones.

1.2 Formal Verification

Formal verification is the task of proving that an that algorithm operates correctly with respect to a formal property. This task takes an algorithm and one or more properties as inputs and returns the answer of either *correct* or *incorrect*. Finding this answer is not trivial. Several techniques and tools have been developed to assist in achieving formal verification.

An example of formal verification can be demonstrated with loop-freedom, a property that is common in routing protocols. A loop occurs when two nodes indefinitely forward a packet between each other. As a result, the packet never actually reaches its intended destination. Loops can occur when a node, *A*, believes it can reach the destination through another node, *B*, and *B* believes it can reach the same destination through *A*, Figure 1.1 depicts this situation. Formal verification can be performed on a routing protocol against the property of loop-freedom. If the protocol is always free of loops, then an ideal formal verification will output *correct* and an associated proof of correctness; otherwise the output is *incorrect* with a counter-example describing a scenario causing the loop. In most cases it is easier to prove that an algorithm is incorrect by providing a counter-example.

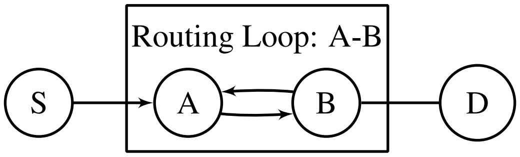


Figure 1.1: A routing loop: *S* sends data destined to *D*; *A* forwards data to *B*; *B* forwards data to *A*; cycle repeats indefinitely and data never reaches *D*.

1.3 Research Problem

A framework is needed for the formal verification of secure MANET routing protocols with respect to their stated security goals. The framework will provide concise protocol, property, and environmental definitions. These definitions will lead to an improved, fact-based discourse of secure routing protocols. An execution environment for the framework will allow the protocols to be examined in the context of a distributed system domain, without the need for physical implementation.

Accuracy and availability are the two goals associated with secure routing. When these goals are achieved, the protocol is said to have the properties of accuracy and availability. A routing protocol is secure if it creates routes that are accurate and those routes are always accurate. When faced with node failures and active attacks, a secure protocol must continue to meet its goals.

1.4 The Formal Verification Framework

Secure MANET routing protocols are complex. This complexity compounds with mobility and security causing all of a verification's possible states, known as *state-space*, to increase exponentially. State-space explosion has made proofs for secure MANET protocols difficult. However, formal methods have shown some promise in revealing counter-examples for protocols and their security goals. Specifically, the Simple Promela Interpreter (SPIN) has been applied to several secure, reactive routing protocols and has automatically discovered scenarios in which security goals fail.

Andel [2] developed a framework around SPIN for exhaustively verifying a subset of secure, accumulation-based, reactive MANET routing protocols against the property of security. The framework is described fully in Chapter 2. Andel's framework is unable to analyze table-based and proactive protocols. This research adapts formal verification and Andel's approach to perform the property verification of table-based and proactive routing protocols. The improved framework is demonstrated for two proactive, table-based routing protocols, but is extensible to all table-based routing protocols.

1.5 Outline

Chapter 2 is a survey of previous works in secure MANET routing verification and presents important background information about SPIN and secure routing protocols. Chapter 3 is the approach taken for the verification and validation of secure, table-based MANET routing protocols. Five example attack vectors are introduced. Chapter 4 provides the validation and verification results for the Optimized Link-State Routing (OLSR) and Secure OLSR protocols against the attack vectors. The conclusions and future work are discussed in Chapter 5.

2 Background and Literature Review

This chapter provides background on the field of wireless routing protocols. It presents the special challenges associated with wireless routing and discusses the verification efforts applied to routing protocols. Representative reactive and proactive protocols are introduced in both insecure and secure implementations. An emphasis is placed on Optimized Link-State Routing (OLSR) and Secure OLSR. The Simple Promela Interpreter (SPIN) verification tool is also introduced, and specific features are related to their applications in mobile ad hoc network (MANET) protocol verification.

2.1 Mobile Ad Hoc Networking

Routing for physically connected devices has two commonly accepted solutions, the link-state and distance-vector routing strategies. For both, a relatively static network, in which the end-systems and routers change infrequently, is assumed. MANETs challenge this assumption because mobile devices are not immobile.

The first wireless network, ALOHANET [3], was developed at Hawaii University. This network laid the foundation for digital wireless technologies. ALOHANET was implemented as a point-to-point architecture with a star topology. One wireless router was used to forward data between two hosts. The problem of mobility was not addressed in ALOHANET because all devices participating in the network were static.

In 1977, Ludwig and Roy [4] proposed *saturation routing* to solve the mobility problem. The saturation routing strategy is commonly known as *flooding*. In saturation routing a connection request is sent by an initiator to its nearest switch. This switch then forwards the request on to all other switches in the network. If the destination exists on a particular switch, that switch identifies itself to the initiator. The switches are responsible for discovering the destination and establishing a virtual circuit connection to it. This solution required routing to be accomplished by specialized hardware with the sole purpose of routing information.

In 1981, Gafni [5] proposed a method for routing data between packet radios. The approach required each device to perform switching functions in addition to its primary radio function. This paradigm diverges from the established infrastructure based approach of ALOHANET and other communication networks of their time. Gafni's approach is considered the first ad hoc routing protocol [6].

The primary function of wireless devices in an ad hoc network varies depending on its goal. In addition, all these devices must fulfill the critical secondary function of routing data within their network. Routing protocols are concerned with the secondary function. Today's ad hoc routing schemes have not changed much from their origins. Flooding, distance-vector, or link-state routing strategies can be found at the core of every ad hoc routing protocol. The ad hoc implementations of these strategies differ from their origins in that they have been optimized to preserve wireless bandwidth, decrease route delay, or decrease power consumption in a wireless environment.

Several well accepted protocols exist for performing the routing function in ad hoc networks. These are Dynamic MANET On-demand Routing (DYMO), Optimized Link-State Routing (OLSR), and Simplified Multicast Forwarding (SMF). Each protocol has been selected by the Internet Engineering Task Force (IETF) MANET Working Group (WG) [1] for standardization.

2.1.1 The Environment.

Ad hoc networks operate in a significantly different environment than traditional networks. Traditional networks depend on access to infrastructural assets. Such assets may be the routers and cabling in an office building or a wireless access point that provides wireless devices connectivity to the Internet. In both examples fixed infrastructure is used to accomplish the goal of connecting end users. An

[1] http://datatracker.ietf.org/wg/manet/charter/

ad hoc network does not have the same reliance on infrastructure. Ad hoc networks are unique because they require every node to act as a router and they operate in a wireless environment with the potential for high-mobility.

The ad hoc environment provides unique challenges that routing protocols must meet. Wireless networks must account for the broadcast nature of the environment, which impacts network density and security. Ad hoc routing protocols must also tolerate high levels of mobility caused by movement, malfunction, failure, or compromise of any set of nodes within the network.

The biggest problem faced in wireless networking is that of broadcast transmission, because all communication can be overheard by unintended recipients. In a wireless network all routing information is broadcast from each participating node. Given this structure, no single node is poised to function as a bastion from which to protect the whole network. The exact footprint depends on the antenna and power of the transmission. It is generally assumed that the transmission power is equivalent for all nodes and that the antennas are omni-directional with comparable gains. Directional antennas have been proposed; however, their implementation is difficult in ad hoc networks because of mobility.

Although the ALOHANET's star topology was simple, it encountered two primary problems: wireless digital transmissions had a high error rate, and simultaneous transmissions on the same channel were lost. This same problem still affects all wireless networks today. Collisions lead to a limitation in the number of users on the wireless medium. Gupta and Kumar [7] demonstrate that the capacity, in bits per second, is upper and lower bounded by $\Theta\left(1/\sqrt{nlog(n)}\right)$, where n represents the number of nodes participating in a wireless network. The capacity of a wireless network approaches zero as the number of users approach infinity. This limitation is a direct result of broadcast transmissions colliding. Carrier detection would prevent packet collisions; however, the wireless environment makes this infeasible and collision avoidance is used instead. Collision avoidance is implemented at the data link layer and is generally not the concern of the routing protocol.

Signal jamming is another problem exposed by wireless networks. Unintentional jamming can be the result of adjacent wireless systems or too many participating nodes. Intentional jamming is easily performed by an adversary broadcasting continuously at a high power level. Jamming, in either form, disrupts communication to some or all nodes in a network and can be considered denial of service.

Node mobility manifests in network topologies that are constantly changing. For infrastructure based networks an ongoing session must not be interrupted even if the user moves out of range of one access point into the range of another in the same network. Handling topological change is only the responsibility of the access points. For ad hoc networks mobility means an ongoing session is not interrupted. If an existing path fails a new path needs to be established to maintain the session. This maintenance may involve reconfiguring multiple hops until a new route is established, and must appear seamless to the users. Mobility is a significant challenge which all MANET routing protocols must address in their implementations.

Discovering routes in a MANET is another challenge. Mobility causes formed routes to break; therefore, addressing mobility is related to route discovery. Depending on the intent of the protocol, some compromises may need to be made in route latency, message overhead, or other properties. These choices can be manipulated to achieve certain design goals, such as quality of service, energy efficiency, and route freshness.

Wireless networks are exposed to many security vulnerabilities. Adversaries have the potential to listen, modify, relay, and inject communication. In an unprotected network, an adversary can subvert the routing process by injecting traffic that does not follow a protocol's specification. While, a protected network is more difficult to attack, a combination of modified and relayed traffic may make it is possible for an outsider to subvert the routing process. In either case, it is important to realize that an adversary may take advantage of certain technologies, such as directional antennas, to improve their ability to listen to and inject traffic. Secure protocols may have subtle, unknown flaws.

The benefits of untethered digital devices far outweigh all the problems associated with the wireless medium and routing. As wireless communication is accepted more and more, it becomes imperative that all of the associated vulnerabilities are understood for operational purposes. Mitigation is possible, but the first step is understanding the risks.

2.1.2 Applications. Ad hoc networks take on different names and meanings based on the specific application. In this research, a MANET is a self configuring, packet switched network, composed of wireless nodes where each node functions as both an end-device and router. Nodes in a MANET are commonly sensors, network bridges, communication devices, databases, or combinations of each.

The term wireless mesh network (WMN) is used to describe a network composed of both MANET nodes and fixed infrastructure, e.g., access points, cell towers, etc. Wireless sensor networks (WSNs) are usually deployed in inhospitable environments and in imprecise configurations with limited power per device. Therefore, it is essential that a sensor network be self-configuring and self-healing. WSNs are the subset of MANET routing protocols devoted to energy aware routing. Sensor networks tend to make assumptions about node density, battery life, and processing power. Vehicular ad hoc networks (VANETs) are specializations of the more general MANET. These networks place assumptions on aspects of node mobility, e.g., speed, and direction of travel. The only assumption placed on the MANET is that all nodes are able to exhibit mobility and must perform the required routing function. MANETs are the focal point of this research because they encompass the goals of WMNs, WSNs, and VANETs.

Some recent literature has focused on specialized versions of MANETs. For example, Stajano [8] and Karlof [9] both discuss security of ad hoc networks, but only address the class of WSN routing protocols. While this focus is important to the specific sensor application of ad hoc networking, it loses sight of the larger class of ad hoc protocols.

2.1.3 Protocols. The underlying technology of MANETs is the routing protocol. Protocols ensure that devices in the network can communicate with one another. There are many ways to classify MANET routing protocols, the three most important of which are *single-phased* or *two-phased*, *accumulation-based* or *table-based*, and *proactive*, *reactive*, or *hybrid*. These classifications characterize the basic operation of any MANET routing protocol. Each attribute carries with it distinctive benefits and drawbacks. For example, a proactive routing protocol exhibits low route latency, but requires more control traffic for establishing routes that may never be used.

The concept of single-phased and two-phased protocols is used by Andel [10]. Single-phased protocols are characterized by data forwarding without the use of pre-established routes. The flooding strategy is classified as a single-phased protocol, as well as any protocols that merely provide optimizations on top of flooding. Two-phased protocols are divided into route discovery and data forwarding phases. Route discovery determines what paths are available in a network. The data forwarding phase sends data through these paths, and thus is dependent on the route discovery phase. If routes become inaccurate then a two-phased protocol should re-initiate route discovery.

The accumulation-based or table-based classification refers to the way a protocol stores routes between nodes. In either case the classification implies a two-phased protocol because routes are discovered before being accumulated or stored in tables. Accumulation-based protocols embed routes in a control segment of packetized data. At each hop the embedded control data is used to accumulate the previous hop. Table-based protocols maintain routing tables in every node. The routing tables at each node are used to make the next hop decision for packets based on the embedded destination.

The proactive, reactive, or hybrid classification relates how a routing protocol performs route discovery. Route discovery implies all protocols under this classification are two-phased. Proactive routing protocols continually build all possible source-to-destination routes within a network. Every node independently stores a route for every destination. Route storage also implies proactive protocols are table-based. Reactive protocols build source-to-destination routes on-demand, caching active routes. Some reactive routing protocols utilize route accumulation, storing routes only at the destination, others are table-based and maintain a forwarding table in every node. In reactive, table-based protocols, route entries are only added to a node's table if the node is in a path between the source and destination. Hybrid protocols combine proactive and reactive behaviors. The regions of proactive and reactive operation are usually separated by a metric such as hop-count [11].

By far, the most useful classification is that of proactive or reactive. This classification is followed in importance by the accumulation-based or table-based classification. Finally single-phased or two-phased classification helps to differentiate the previous two classifications from flooding strategies. The two-phased classification provides a dichotomy between route discovery and data forwarding. The remainder of the routing discussion is limited to the route discovery phase. Routing security is only concerned with route discovery. If route discovery provides inaccurate results then the data forwarding phase is negatively impacted.

Designing a routing protocol is an art of balancing requirements and desirable features. An illustrative example is the balance between route latency, update period, and message overhead. Route latency refers to the time between when a route is requested for transmission and when it becomes available. Update period refers to how often routes are refreshed and reflects on how well the protocol can handle mobility. *Reactive* protocols feature high latency because routes are discovered only when needed so a route is guaranteed but takes more time to form. *Proactive* protocols feature low latency because routes are formed continually, but routes are not guaranteed because mobility may invalidate a route between updates. The trade-off between latency and route availability is a common example of choices that must be considered in the design of a protocol. In accumulation-based protocols message overhead is added to the transmitted data. Table-based protocols use separate control messages for route discovery. Other trade-offs are also possible for achieving goals, such as, energy conservation, and security.

There are many MANET routing protocols. A few appear in Table 2.1, where each protocol's respective classification is briefly captured. Notably, Dynamic MANET On-demand Routing (DYMO) and Optimized Link-State Routing (OLSR) were both selected by the IETF MANET WG for standardization.

Table 2.1: Selected MANET routing protocols.

Protocol	**Classification**
Dynamic source routing (DSR) [12]	Reactive, Accumulation
Dynamic MANET On-demand Routing (DYMO) [13]	Reactive, Table-based
Ad hoc On-demand Distance Vector (AODV) [14]	Reactive, Table-based
Lightweight Underlay Network for Ad hoc Routing (LUNAR) [15]	Reactive, Table-based
Optimized Link-State Routing (OLSR) [16]	Proactive, Table-based
Destination Sequence Distance Vector (DSDV) [17]	Proactive, Table-based

2.1.3.1 Reactive, Accumulation-based. Dynamic source routing (DSR) [12] is a very simple ad hoc routing protocol. The protocol is two-phased and its route-discovery phase can be described as a flooded route request and directed route reply from the destination node. The routes are accumulated by the request and forwarded in the reply. Every request received at the destination triggers a reply message. For example, if node *A* sends a route request (RREQ) message destined for *C*, then node *B* forwards the RREQ to node *C*. Since node *C* is the destination of the request, node *C* sends a route reply (RREP) message with the accumulated route, *C*-*B*-*A*. Node *A* eventually learns of the *A*-*B*-*C* route to *C* and uses this route until it expires or breaks.

2.1.3.2 Reactive, Table-based. Ad hoc On-demand Distance Vector (AODV) [14] and Dynamic MANET On-demand Routing (DYMO) [13] are reactive, table-based protocols. AODV and DYMO are loosely based on the fundamental RREQ/RREP design of DSR with the exception that the intermediary nodes maintain the routes rather than forwarding each route through the network. Both protocols are table-based. As a RREQ is forwarded through the network a reverse path, to the sending node, is recorded at each intermediary node. When the RREQ reaches the destination, a RREP message is forwarded along the reverse path. As the RREP is propagating through the network each node records the sender of the reply, thus setting up a reverse path to the destination. The route discovery phase is complete upon receipt of the RREP at the requesting node. The DYMO protocol is a simpler version of AODV which removes unnecessary route discovery procedures. Both protocols fundamentally operate in the same way.

2.1.3.3 Proactive, Table-based. Optimized Link-State Routing (OLSR) [16] relies on the link-state routing strategy in which every node learns the neighbors of every other node in the network. OLSR uses multi-point relays (MPRs) to improve performance by decreasing the number of broadcasting nodes. MPRs are responsible for forwarding data on behalf of other nodes. Only MPRs can forward packets, which limits the number of nodes broadcasting and minimizes the amount of flooding in the network. OLSR is proactive so it is continually performing the route discovery phase of the two-phased process.

The OLSR protocol is of particular interest in this research because it is both proactive and table-based. In the context of security and verification only reactive, accumulation-based protocols have been thoroughly studied, a few researchers have examined reactive, table-based protocols, e.g., Wibling [15] examines LUNAR, but there is no similar literature for proactive, table-based routing protocols. OLSR is used as a case study for this research, and for this reason the details of the protocol are thoroughly examined in the remaining part of this section.

At the core of the Optimized Link-State Routing (OLSR) protocol's function is the definition of *neighbors*, *two-hop neighbors*, and *multi-point relay (MPR)* sets. A neighbor is defined as a node within transceiver range of another node. Two-hop neighbors are the neighbors of a neighbor. A multi-point relay (MPR) set is a set of neighbors requesting the current node to relay data to its neighbors. The MPR set provides a path from a node to its two-hop neighbors. A goal in OLSR is to distribute MPR sets to all nodes in the network. If every node has accurate and complete MPR and neighbor sets then optimal route selection is possible between every node. OLSR's operation can be divided into two general processes: *neighbor sensing* and *topology control*, both run concurrently. A third process, *MPR Selection*, is the process of selecting relaying nodes and is part of the neighbor sensing process. Each node maintains internal state variables for these three processes as shown in Table 2.2.

During neighbor sensing HELLO messages are generated by all participating nodes. HELLO messages traverse only one wireless link, i.e., a single hop. Table 2.3 contains the information that every node uses in a HELLO message. A node which has not discovered any neighbors will periodically transmit an empty HELLO message. When a HELLO message is received the information it contains is added to the receiving node's Neighbor

Table 2.2: OLSR Fields Maintained by Each Node

Field	**Purpose**
Neighbor Sensing	Data from HELLO messages
Neighbor Set	One-hop neighbors
Two-hop Neighbors	Two-hop neighbors
MPR Set	Nodes to relay for
MPR Selection Set	Nodes selected as MPRs
MPR Selectees	Nodes selected as relays
ANSN	Freshness of MPR Set
Topology Control	Topology related data
Origin	Reachable destination
MPR Selectors	Nodes selecting origin as relay
ANSN	freshest MPR data

Set, shown in Table 2.2. If this information changes a node's Neighbor Set, then a new HELLO message is transmitted by the receiving node. Eventually, through HELLO messaging, every node learns of its neighbors and two-hop neighbors. Each node selects which of its neighbors should forward data on its behalf, which is the MPR selection process. Subsequent HELLO messages inform a node if it has been selected using the MPR Selection Set field of the HELLO message.

Table 2.3: OLSR HELLO Message Format

Field	**Purpose**
Origin	HELLO originator
Neighbors	One-hop neighbors
MPR Selection Set	Neighbors selected as MPRs

Shorthand: <{N},{MPR},Source>

The shorthand in Table 2.3, < {N}, {MPR},Source>, provides the format of the message used in the abstracted model. Relative to the *Source*, {N} is the set of neighbors and {MPR} is the set of selected MPRs. This notation is used in Figure 2.2.

During topology control, a node selected as an MPR broadcasts a topology control (TC) message with the structure described in Table 2.4. The *MPR* field contains the identifiers of all nodes which have selected the *Origin* (i.e., the node generating the TC message) as a relay. The *advertised neighbor sequence number (ANSN)* field is used to determine the freshness of a TC message. Nodes will only use a TC message if the *ANSN* is greater than that of any previously received *ANSN* for the same *Origin*. If the time to live (TTL) is greater than zero, then any node with a non-empty *MPR Selector Set* must forward a received TC message after decrementing its TTL. Using this forwarding rule TC messages spread through the whole network if the initial *TTL* is large enough (i.e., greater than the number of nodes in the network). Finally, the *source* field denotes the node from which the TC message was last received. As a matter of style, *topology control* always refers to the process and TC always refers to the message.

Table 2.4: OLSR Topology Control (TC) Message Format

Field	**Purpose**
Source	Last node to forward TC
Origin	TC message originator
MPRs of Source	Selected MPRs of origin
ANSN	Maintain freshest MPR
Time To Live (TTL)	Limits TC life-time

Shorthand: < {s}, ANSN, TTL,Source,Origin>

The notation < {s}, ANSN, TTL,Source,Origin> below Table 2.4 is shorthand for the data exchanged in the abstracted TC messages. The {s} field is the set of nodes which have selected *Origin* as a mutlipoint relay. This shorthand appears in Figure 2.2.

MPR selection is triggered whenever a node receives new information from a HELLO message. The goal of the selection process is for the selector to maintain a minimal set of neighbors which guarantees all two-hop neighbors are reachable. If the selection results in a change to the node's MPR Selectees, then subsequent HELLO messages are sent with the changes. Upon a node receiving a HELLO message, if the message's MPR Selection Set contains the current node's identifier, then the node adds the message's Origin to its Topology Control Origin field. This last step alerts a node that it has been selected as an MPR.

Minimal MPR selection is important in respecting the two most important wireless resources: the transmission medium and battery power. If every neighbor is always selected as an MPR, then repeated collisions occur on the transmission medium and every node wastes energy in transmitting redundant messages. This complete selection is no better than simple flooding, but minimal MPR selection is NP-complete [18]. There is no known polynomial time algorithm for computing a minimal MPR set. Polynomial time algorithms can only provide estimates of the minimal set, known as heuristics.

For the purposes of this research, the default MPR selection heuristic of OLSR is employed. Several other heuristics are available, but this particular one is prescribed in RFC 3626. Listing 2.1 presents the heuristic. Any MPR selection strategy could be employed but the research goal is to analyze security properties rather than protocol efficiency.

A shortest path algorithm computes the routing table using the information obtained through topology control. Topology control need not be complete to run the algorithm, but until it is complete the shortest path is not necessarily available. Assuming a static topology, the routing table eventually becomes complete for every node in the network. The routing table is maintained as a standard link-state routing table.

An example of the OLSR protocol in operation is captured in Figure 2.2. The scenario refers to the physical topology in Figure 2.1 where node v_0 is in range of v_1 and v_1 is in range of v_2. The nodes in OLSR operate asynchronously, therefore the exact ordering of events shown is only one possible ordering. For more details on OLSR's behavior refer to RFC 3626 [16].

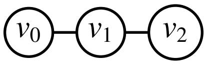


Figure 2.1: Simple three node topology, T=5.

Listing 2.1: MPR Selection Algorithm [16].

```
set N2[n], neighbor;

set MPR_Selection( set ans, int src)
  selectee = ∅; /* (i) Remove all entries from selectee */
  ans = ans \ id;  /* (ii) Exclude yourself from MPR (selectee) */

  N2[r_src] = N2[src] | ans; /* Maintain N2[src] */
  set myN2 = ∅;

  /* (iii) Exclude all N from MPR -- neighbor may change on execution */
  foreach($i \in N$)
    N2[i] = N2[i] \ neighbor; /* Eliminate unneeded 2-hops */

  foreach($i \in N$) /*(3) add all i in N with a unique link */
    if(|(N2[i] \ $\bigcup_{j \neq i}$ (N2[j]))| > 0)
      selectee = selectee ∪ i;

  foreach($i \in N$)
    if($i \in selectee$)
      myN2 = myN2 ∪ N2[i];

  while(|($\bigcup_i^N$ N2[i]) \ myN2| > 0) /*(4) Cover all N2 */
    int maxI = 0; /* (4.2) MPR Selection Heuristic */
    int maxV = |N2[0]|;
    forloop(j = 1; j >= n-1; j++)
      if(|N2[j]| > maxV)
        maxI = j;
        maxV = |N2[j]|;
    selectee = selectee ∪ maxI;
    myN2 = myN2 ∪ N2[maxI];
```

To keep the walk-through of Figure 2.2 as concise as possible, several abbreviations and conventions are adopted. Most notation conforms to standard set notation but there are some exceptions:

- The subscript *i* in $HELLO_i$ and TC_i is strictly for clarity, in OLSR messages are distinguished only by their contents and transmission order.
- The symbol *Rx msg* means the current node receives the message *msg*.
- The symbol *Tx msg* means the current node transmits message *msg*.

- The symbol → reads, “it follows that.”
- The symbol v_i.S represents the MPR Selector Set of node v_i.
- The symbol v_i.MPR represents the MPR Set of node v_i.
- The symbol v_i.N represents the neighbors of node v_i.
- The symbol $v_i.\text{N}_2$ represents the two-hop neighbors of node v_i.
- The symbol *id* represents the address or identifier of the current node.
- The symbol $\text{HELLO}_i.src$ represents the origin of the message HELLO_i.
- The symbol v_i.TC represents all topology control information known by v_i.
- The statement *x changed* (*x unchanged*) means elements were added or removed from set *x* (set *x* is unchanged).
- The statement *ignore x* means that element *x* is not considered. The statement usually appears because the element, *x*, is the address of the current node.

Upon completion of neighbor sensing, topology control builds complete knowledge of MPRs throughout the network for each node. Topology control runs concurrently with neighbor sensing, but both processes are separated in Figure 2.2 for clarity. An OLSR model must consider the concurrent exchange of HELLO and TC messages.

Neighbor sensing requires the exchange of six HELLO messages and topology control requires only one TC message for the small three-node network depicted in Figure 2.1. When considering all possible concurrent exchanges of TC and HELLO messages the number of possible states explodes leading to the difficulty of formal verification.

2.2 MANET Security

Traditionally, security entails the need of providing confidentiality, integrity, and availability (CIA). Providing confidentiality and integrity in the layers above the network layer has been heavily researched. Many methods for providing secure services are widely accepted. For example, Public Key Infrastructure (PKI) is a solution that provides confidentiality, integrity, and non-repudiation on user data. Despite the relevance of

HELLO Messaging

Node v_0: $v_0.\text{N} = \emptyset$; Tx $\text{HELLO}_1 :< \emptyset, \emptyset, v_0 >$

Node v_1: Rx HELLO_1; $\text{HELLO}_1.src \notin \text{N} \rightarrow v_0 \cup \text{N}$; $\text{HELLO}_1.\text{N} = \emptyset \rightarrow \text{N}_2$ not changed → no changes to $v_1.\text{MPR}$ or $v_1.\text{S}$; $v_1.\text{N}$ changed → Tx HELLO_2: $< \{v_0\}, \emptyset, v_1 >$

Node v_0: Rx HELLO_2; $\text{HELLO}_2.src \notin v_0.\text{N} \rightarrow v_1 \cup v_0.\text{N}$; $\text{HELLO}_2.\text{N} = \{v_0\}$, id=$v_0 \rightarrow v_0.\text{N}_2$ not changed → no $v_0.\text{MPR}$ changes; $\text{HELLO}_2.\text{MPR} = \emptyset \rightarrow$ no changes to $v_0.\text{S}$; $v_0.\text{N}$ changed → Tx HELLO_3: $< \{v_1\}, \emptyset, v_0 >$

Node v_2: Rx HELLO_2; $\text{HELLO}_2.src \notin v_2.\text{N} \rightarrow v_1 \cup v_2.\text{N}$; $\text{HELLO}_2.\text{N} = \{v_0\}$, id=$v_2 \neq v_0 \rightarrow v_0 \cup v_2.\text{N}_2$; $v_1 \cup v_2.\text{MPR}$ (reach-ability to v_0); $\text{HELLO}_2.\text{MPR} = \emptyset \rightarrow$ no change to $v_2.\text{S}$; $v_2.\text{N}$ and $v_2.\text{MPR}$ changed → Tx HELLO_4: $< \{v_1\}, \{v_1\}, v_2 >$

Node v_1: Rx HELLO_3; $\text{HELLO}_3.src \in v_1.\text{N} \rightarrow v_1.\text{N}$ is unchanged; $\text{HELLO}_3.\text{N} = \{v_1\}$, id=$v_1 \rightarrow v_1.\text{N}_2$ not changed → $v_1.\text{MPR}$ not changed; $\text{HELLO}_3.\text{MPR} = \emptyset \rightarrow v_1.\text{S}$ not changed; v_1 enters idle state

Node v_1: Rx HELLO_4; $\text{HELLO}_4.src \notin v_1.\text{N} \rightarrow v_2 \cup v_1.\text{N}$; $\text{HELLO}_4.\text{N} = \{v_1\}$, id=$v_1 \rightarrow v_1.\text{MPR}$ not changed; $\text{HELLO}_4.\text{MPR} = \{v_1\} \rightarrow v_1 \cup v_1.\text{S}$; $v_1.\text{N}$ changed → Tx HELLO_5: $< \{v_0, v_2\}, \emptyset, v_1 >$

Node v_2: Rx HELLO_5; $\text{HELLO}_5.src \in v_2.\text{N}$, $v_2.\text{N}$ is unchanged; $\text{HELLO}_5.\text{N} = \{v_0, v_2\}$, id=$v_2$, ignore v_2; $v_0 \cup v_2.\text{N}_2$, $v_2.\text{N}_2$ remains unchanged; v_2 enters idle state

Node v_0: Rx HELLO_5; $\text{HELLO}_5.src \in v_0.\text{N}$, $v_0.\text{N}$ is unchanged; $\text{HELLO}_5.\text{N} = \{v_0, v_2\}$, id=$v_0$, ignore v_0; $v_2 \cup v_0.\text{N}_2$; $v_1 \cup v_0.\text{MPR}$; $\text{HELLO}_5.\text{MPR} = \emptyset \rightarrow$ no change to $v_0.\text{S}$; $v_0.\text{MPR}$ changed → Tx HELLO_6: $< \{v_1\}, \{v_1\}, v_0 >$

Node v_1: Rx HELLO_6; $\text{HELLO}_6.src \in v_1.\text{N}$, $v_1.\text{N}$ is unchanged; $\text{HELLO}_6.\text{N} = \{v_1\}$, id=$v_1$, $v_1.\text{MPR}$ not changed; $\text{HELLO}_6.\text{MPR} = \{v_1\}$, $v_0 \cup v_1.\text{S}$; $v_1.\text{MPR}$ and $v_1.\text{N}$ are unchanged; v_1 enters idle state

Topology Control

Node v_1: $v_1.\text{S} = \{v_0, v_2\} \rightarrow$ Tx TC_1: $< \{v_0, v_2\}, v_1 >$

Node v_0: Rx TC_1; $v_0.\text{TC} = \emptyset \rightarrow \{v_0, v_2\} \cup v_0.\text{TC}$

Node v_2: Rx TC_1; $v_2.\text{TC} = \emptyset \rightarrow \{v_0, v_2\} \cup v_2.\text{TC}$

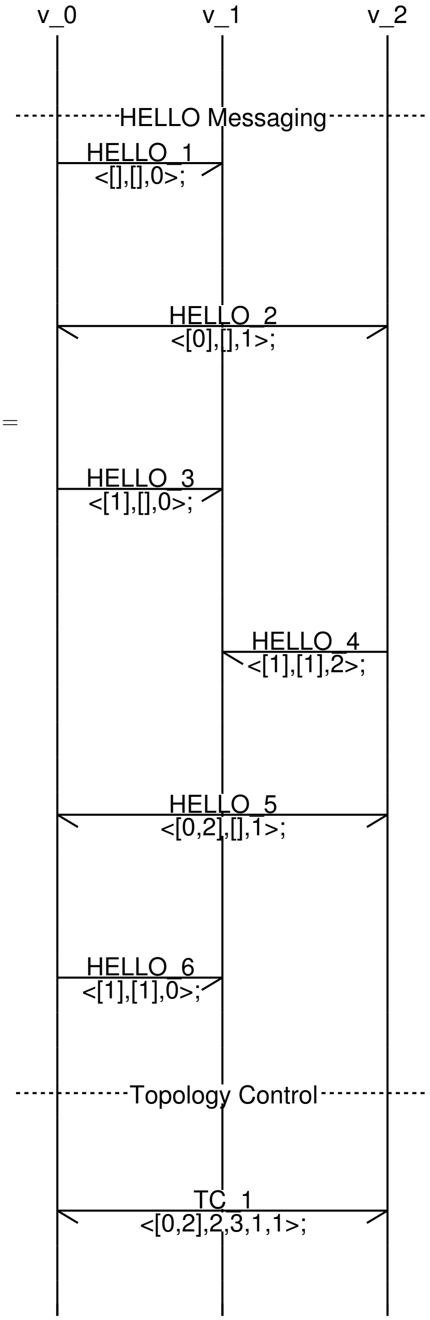


Figure 2.2: Message sequence chart of an OLSR execution under topology, T=5.

confidentiality and integrity, nothing can rival the importance of providing availability. Stajano [8] points out that little else matters if the required system is not available. Routing information is significantly different than data because it cannot be encrypted and still remain useful. Routing security can be thought of as the requirement to keep network paths available for communication. Andel [2] provides a more precise definition of security:

Accuracy: A routing protocol is accurate if it produces routes that meet its objectives.

Reliability: A routing protocol is reliable if its returned routes are always accurate.

Security: A routing protocol provides security if it preservers the protocol's accuracy and reliability in the presence of malicious attackers.

The objective is providing end-to-end communication between two nodes over existing wireless links. This refined definition of security provides a solid foundation for security analysis of all routing protocols. Secure protocols use integrity as a tool for providing security. If the integrity of the route discovery process can be assured then the protocol must be considered secure.

In examining any network related service it is valuable to relate the services to the Open Systems Interconnection model (OSI model). Table 2.5 presents which layers are responsible for securing routing information versus data. This is helpful in limiting the layers that need to be tested. Each layer builds its capabilities based on the ones below it. The separation keeps each layer from needlessly complicating the previous layers. However, the weaknesses of the lower layers propagate into the upper layers. For example, the physical layer of IEEE 802.11-2007 is easily jammed with interference, and jamming cannot be overcome anywhere above the physical layer.

Availability is the responsibility of each layer in the OSI model; however, for the study of MANET routing protocols availability is limited to the physical, data link, and

Table 2.5: The Open Systems Interconnection model (OSI model) and security.

Application Layer Presentation Layer Session Layer Transport Layer	Secure Data
Network Layer Data Link Layer Physical Layer	Secure Routing

network layers. At the physical layer it is possible for availability to denied by signal jamming, interference, or congestion. The data link layer can lose availability if a device subverts the medium access control process. For example, a node that continuously contends for medium access without waiting the short inter-frame spacing period subverts the IEEE 802.11-2007's medium access control. The only difference from interference jamming is that the medium access denial of service participates in the medium access control process. Subverting the network layer can occur if a malicious node inserts itself and controls a link in an active route. The malicious node then has the ability to manipulate data at any point. For example, the node could drop data. It is not safe to assume that any layer is inherently secure. The examples provided for the lower three layers show that each is susceptible to simple attacks. An unavailable network is of no use.

Table 2.6 lists several secure routing protocols with their respective classifications. Only protocols designed with the goal of security are included. All of the protocols listed in Table 2.1 have limitations to their security. The broadcast nature of wireless signals gives attackers the ability to listen to all network traffic. It is possible for an attacker to inject information that may cause a protocol to make inaccurate routing decisions. The protocols listed here have been created to prevent this type of attack, but not all of them provide unconditional protection. The protocols are assumed to function at the network layer of the OSI model.

Table 2.6: Pertinent Secure MANET routing protocols.

Protocol	**Classification**
endairA [19]	Reactive, Accumulation
Ariadne [20]	Reactive, Accumulation
Secure Efficient Distance Vector (SEAD) [21]	Reactive, Table-based
Authenticated Routing for Ad Hoc Networks (ARAN) [22]	Reactive, Table-based
Secure AODV (SAODV) [23]	Reactive, Table-based
Secure Link State Routing Protocol (SLSP) [24]	Proactive, Table-based
Secure OLSR [25]	Proactive, Table-based

2.2.1 Secure, Reactive, Accumulation-based Protocols. Ariadne [20] and endairA [19] are both secure, reactive, accumulation-based protocols. The routes in both protocols contain route information that is constantly changing during a RREQ and RREP. These changes make it difficult to detect if any routing traffic has been modified or falsely inserted. Both protocols include signed hashes on every modification to prove that only trusted nodes are participating in route formation. The endairA protocol adds authentication to RREQ messages as well. Ariadne and endairA have been analyzed for their security properties by Ács [19], Andel [2], and Benetti [26].

2.2.2 Secure, Reactive, Table-based Protocols. Secure AODV (SAODV) [23] incorporates digital signatures for authenticating immutable data fields and hash-chains for securing the hop-count field. In addition, route error messages are digitally signed and must be verified before being used. See Appendix B.1 for a detailed description.

The Secure Efficient Distance Vector (SEAD) [21] protocol guarantees an upper bounded sequence number and a lower bounded hop count of each route. The upper bounded sequence number ensures that an attacker cannot force destination nodes to perform an arbitrarily large set of hash operations. The lower bounded hop count ensures that an attacker cannot advertise a route better than the one received. Bounding on hop-count and sequence number is accomplished through one-way hash chains. Appendix B.2 describes SEAD in depth.

2.2.3 Secure, Proactive, Table-based Protocols. Secure Link State Routing Protocol (SLSP) secures link-state updates and HELLO messages with public key cryptography. The link-state updates are similar to the topology control messages used in OLSR, although SLSP does not include any of the optimizations proposed by OLSR. Appendix B.3 contains a description of SLSP.

Adjih [25] proposes a strategy for securing OLSR. This strategy, referred to as Secure OLSR, involves the use of public key cryptography and timestamps. The author claims the design can be extended to any proactive routing protocols. Secure OLSR is a natural extension to the OLSR routing protocol, and is used as a case study for secure, proactive, table-based routing protocols. A detailed description of Secure OLSR follows.

Secure OLSR [25] only differs from OLSR in that two functions are added and a new message type is proposed. The two functions are *verif()* and *sign()*. The function *sign()* produces a signature hash of an original message and the originator's private key. The *verif()* function checks that a message corresponds to its signature given the public key of the originating node. A new message type is defined for sending the signature to each node. When a control message (TC or HELLO) is received no action is taken until the corresponding signature is received and this signature authenticates the message. In this way Secure OLSR can exclude nodes from the network that do not possess the correct private keys.

Secure OLSR [25] aims to prevent unauthorized nodes from participating in the network. The protocol assumes that public keys are securely distributed to all participating nodes. In this research, it is assumed that public keys are known at each node and the private keys for each node are known only to the respective nodes, unless otherwise noted. Secure OLSR claims to prevent outsiders from injecting inaccurate routes into the network.

Routing attacks are prevented by using a signature mechanism to provide a secure and unique hash of each message. The signing mechanism, *sign()*, is represented by Equation 2.1. The original message is sent in the clear along with the result of *sign()*. These messages may be sent in two separate messages. Each node in the network stores the received messages for a short period until the associated signature is received. If the received signature can be decoded with the sender's public key then the message is processed in accordance with the OLSR protocol. The decoding function, *verif()*, is denoted by Equation 2.2. Any node, for which a public key is not held, is prevented from directly adding traffic to the network. The control messages are always sent in the clear, making it easier for MPR nodes to update the TTL and ANSN fields of TC messages. Since the TTL and ANSN fields are mutable they are set to the constant value, zero, before *sign()* or *verif()* is applied.

$$sign(id, k_{id}^{-}, message) \rightarrow uniquesignature \tag{2.1}$$

$$verif(origin, k_{origin}^{+}, message) \rightarrow \{valid, invalid\} \tag{2.2}$$

2.3 Formal Verification

The goal of verification is to prove that a protocol operates correctly with respect to one or more stated properties. Properties express the operation of an algorithm usually in terms of some output or state. Two common examples related to routing protocol algorithms are loop-freedom and secure route creation. A survey of the verification techniques commonly applied to MANET routing protocols is available in [10].

A verification method must provide some way of modeling an algorithm and a systematic way of either proving or disproving the desired properties of the algorithm. The task of verification ultimately should result in a proof of correctness or a counter-example for the algorithm and property. Methods for finding the counter-example are known as model checkers, those for finding proofs are referred to as theorem provers. In performing the verification of MANET routing protocols it is far more common to find model checkers.

Model checkers exhaustively search a model of an algorithm for property violations and provide counter-examples if property violations are found. This method analyzes all possible states of the system and can be accomplished by a mathematical construct or a computer program. The algorithm's model must accurately represent the behavior of the system. The counter-example is the set of events leading to a property's failure. The abstraction required for modeling algorithms is dependent on the model checker, and usually is a straightforward translation from an algorithm's procedure.

Theorem provers attempt to provide a proof explaining why a particular property holds in an algorithm. The proofs provided can be cryptic because the correctness of the algorithms and their associated properties are modeled in complex, mathematical abstractions. When using theorem provers the algorithms and properties must be put into a mathematical form that the prover can interpret. This transformation generally requires modeling the algorithm in abstract mathematical terms.

2.3.1 Verification Methods. The verification of routing protocols has been performed in many different ways. Andel [10] presents the categories of non-exhaustive and exhaustive verification methods. Non-exhaustive verification methods encompass visual inspection and performance oriented modeling, or simulation. Exhaustive methods include analytical proofs, simulatability models and formal methods.

Simulation relies on tools such as Network Simulator 2 (NS2), OPNET$^{©}$, Global Mobile Simulator (GloMoSim), and NetSIM. Simulation falls short of actively proving that a distributed system will perform correctly or as expected. These tools provide statistical data that enable the comparison of two protocols. An example: protocol X reliably delivers more packets than protocol Y, therefore X is more secure than Y. This statement is not necessarily true because X might deliver zero packets when a discrete attack or error occurs. In lieu of protocol correctness, most researchers provide extensive performance analysis through network simulation.

Visual inspection is another commonly used method. In this method, a protocol is published and attacks and errors are later published against it. Then, a new modified version of the protocol is published to counter such attacks and errors. Such methods are prone to oversight because the technique is solely based in human intuition and understanding of the routing protocols.

Analytical proofs require experienced mathematicians familiar with routing protocols. Simulatability models require mapping protocols into complex mathematical formulations, requiring assumptions about a protocol's operation that may exclude possible behaviors or attacks in the real protocol. Formal methods come in a variety of forms from the hand-based modeling of strand-spaces to computer-based model checking. Formal methods seek to represent a system accurately and provide exhaustive searches of the system for counter-examples; however, not all formal methods can provide a counter example when a violation is found.

The set of tools available in formal methods include strand-spaces, the Simple Promela Interpreter (SPIN), Automated Validation of Internet Security Protocols and Applications (AVISPA), Cryptographic Protocol Analysis Language - Evaluation System (CPAL-ES), process calculus, and petri nets. All of these methods have been applied in some form to secure ad hoc routing verification. Some of the tools are exclusively designed for verifying cryptographic security protocols but have been cajoled into the verification of MANETs. Newer methods for ad hoc network verification have recently been developed in the form of process calculi. Singh [27] proposes the ω-calculus and applies it to verify a property of an ad hoc leader-election protocol. Buttyán and Thong [28] form their own calculus for the verification of secure, reactive, accumulation-based routing protocols. Their calculus can be formalized using the ProVerif tool and used to automatically verify security properties of reactive ad hoc routing protocols.

For the analysis of a secure protocol an attacker model must be developed that provides specific capabilities. Andel [2] proposes the concept of applying a Dolev-Yao [29] attacker. This attack model gives adversarial nodes the ability to listen, inject, modify, replace, or create routing packets. This adversarial model can be thought of as a wiretap into a wireless network. Such an adversary may choose to participate in the routing protocol. A successful Dolev-Yao attacker might be able to advertise a shortest path between a source and destination, thus controlling traffic over that path. Ultimately, finding attacks against secure protocols requires accurately modeling the protocol and thoroughly understanding what an attacker may need to violate security.

2.3.2 Applying Formal Verification to MANETs. *Process calculus* to refers to the whole class of calculi that have been derived from the theoretical work initially performed by Turing [30]. Turing developed what is commonly known as λ-calculus, inspiring the creation of many other calculi. π-calculus was designed to aid in the verification of cryptographic protocols. More recently Singh [27] developed the ω-calculus which can be used to model mobility of nodes in an ad hoc network. This is novel and extremely beneficial to the field based in process calculus. Buttyán and Thong [28] have extended Singh's work by creating a process calculus capable of modeling mobility in addition to the concept of *bi-similarity* which can be used to verify security properties of reactive ad hoc routing protocols.

Andel's definition of security agrees with that of Buttyán and Thong [28] who compare an ideally secure network to a network with unknown security properties using bi-similarity. The basis of their research is to show that an ideal protocol forms a network that is indistinguishable from the one being analyzed. Buttyán and Thong develop a process calculus to support their analysis based on the ω-calculus [27] and π-calculus used for analyzing cryptographic protocols. This methodology requires the security property, in this case the bi-similarity of routes, and the protocols to be modeled within the process calculus. The protocols analyzed are the Secure Routing Protocol (SRP) and Ariadne.

Also, using a bi-similarity, Ács [19] discovered that the Ariadne routing protocol is actually insecure. Ács provides two exemplary attacks against Ariadne that show an adversary advertising a false route which the protocol accepts. Andel [2] is able to automate the process of discovering this attack on Ariadne. Andel's approach replaces the process calculus with a model checker capable of exhaustively searching for security violations, expressed as properties. The attacks analyzed do not consider node mobility, and are effective only in specific topological configurations.

SPIN [31] is a widely used logic model checker, i.e., a verification system. The tool is designed to verify the correctness of distributed systems where several processes are executing concurrently. The Process Metalanguage (Promela) is used to define models in SPIN. Processes are the primary abstraction and each process runs concurrently with all others allowing interaction in interesting and often unexpected ways. For a given Promela model, SPIN is able to exhaustively explore all possible combinations of concurrent interaction.

Promela processes can model the behavior of ad hoc nodes. SPIN enables the concurrent execution of these node models. Andel [2] and Wibling [15] have applied SPIN to the formal verification of ad hoc routing protocols. Wibling [15] performs modeling in Promela that includes transitions between topological network configurations. This approach results in state-space explosion and keeps Wibling from completing an exhaustive search of security properties on all possible network topologies. Andel [2] improves upon this SPIN modeling by treating every possible network topology as a unique verification task. For example, given a five node bi-directional network there are

1,024 different verification tasks. This technique mitigates the state-space explosion problem experienced by Wibling. So far, the approach in Andel [2] applies only to reactive, accumulation-based routing protocols.

The AVISPA [2] project, short for *automated validation of Internet security protocols and applications*, provides a framework for protocol validation via separately maintained tools. The validation portion of AVISPA consists of the tools Constraint Logic-Attack Searcher (CL-AtSe), On-the-Fly Model Cheker (OFMC), and SAT-based Model-Checker (SATMC) for performing model checking. The AVISPA project dates back to 2003 and has been significantly utilized by researchers in the research field of Internet-related cryptographic protocols. AVISPA is currently being replaced with the Automated Validation of Trust and Security of Service-oriented Architectures (AVANTSSAR) project.

AVISPA is similar to SPIN in that modeling is performed through processes. Benetti [26] applies AVISPA to the ARAN and endairA protocols. In this application Benetti formalizes specific scenarios from which an attack is revealed. Pura [32] presents the formal verification of the ARAN protocol with the AVISPA framework. The method in Pura [32] performs the entirety of the model checking in the verifier, to include topological transitions. This approach failed to produce results even for three node topologies. This failure was caused by an exponential increase in verifiable states of the model. Pura claims the AVISPA framework is easier than SPIN. More research is needed to further explore the possibility of using AVISPA in formal routing verification. Benetti's verification of endairA and the ARAN protocols with AVISPA reveal vulnerabilities against the invisible node attack. However the verification was performed with only two topologies and fails to exhaustively explore the search space.

[2] http://www.avispa-project.org/

2.3.3 Promela for MANET Verification. SPIN is selected as the tool of choice for modeling proactive and table-based routing protocols. See Section 3.3.2 for the justification of this decision. An overview of SPIN and Promela's features are now provided in the context of formal MANET routing verification.

SPIN utilizes several faculties for mitigating state-space explosion. The most important is design abstraction. It is not possible, or desirable, to model low level aspects of a system in Promela. A model is best designed by considering the properties of interest and then building the model to verify these properties. Aside from design abstraction, SPIN also takes advantage of state reduction and compression as described in Holzmann [31]. Some of the techniques described are COLLAPSE, for state reduction, and DMA and BITSTATE hashing for state-space compression.

In a Promela model, limiting the number of processes, number of interactions, and memory required for each state decreases the overall state-space. Unfortunately, this requires a trade-off between model detail and model abstraction. The best models in Promela are also the most abstract; they reflect the semantics of the underlying protocol. Holzmann [31] encourages modelers to create the most abstract model first and to then perform verification with the model. If the verification passes (no model property violations are discovered), the modeler should add more detail and re-verify with the updated model. This process continues until either an error is found or the modeler is satisfied that the distributed system has been described in sufficient detail.

Promela and SPIN have several key features that are useful when building ad hoc routing protocol models. At a high level the features can be divided into the categories of correctness claims, state reduction, message passing, control structures, and pre-processors. These key features are roughly presented in this order.

The *never claim* is an omnipotent process in SPIN which is executed after every transition in a verification run. This process is able to observe all global and local variables of each *proctype* using *remoterefs*. The never claim is strictly side-effect free. The never claim provides Linear temporal logic (LTL), allowing logical properties to be tested over time. A simpler correctness claim within SPIN is the assertion which functions just like a C-style assertion.

There are many ways to decrease the size and number of states needed for verification runs. Variables declared using the *hidden* keyword prevent the verifier from tracking their state. Resetting variables to zero tends to reduce the amount of information a verification run must store. SPIN's verification engine has a built-in partial order reduction algorithm which reduces the number of states and the size of those states that must be verified. This reduction is employed by default in any SPIN verification and may be augmented with other reductions such as COLLAPSE and DMA hashing, which both compress the sate-vector, and BITSATE hashing, providing an estimate of exhaustive verification. Unfortunately, the use of *remoterefs* is incompatible with *never* claims and requires SPIN to forgo the standard partial order reduction techniques by setting verifier's the NOREDUCE compiler flag.

Creating deterministic blocks of Promela code, also known as *d_step*, is a very powerful and tricky tool for reducing verification state. The *d_step* keyword in SPIN allows sequences of code to execute as indivisible operations. There are two effects of using *d_step*{...}. The code between the brackets runs deterministically and, more importantly, the transitions within the code are hidden during verification. The hidden transitions lead to a huge reduction in the size of the state-space that must be analyzed. When using *d_step*, jumping in or out of the sequence and message passing is prohibited by the compiler. Sequences are eligible for *d_step* only if they are self contained, but are allowed to read and write local variables.

In Promela the datatype *chan*, short for channel, provides the implementation of message passing. *Send*, represented by *!*, and *receive*, represented by *?*, are two operations defined on a channel. The send and receive functions define communication strictly between two processes. The easiest way to understand a Promela channel is to think of it as a queue that can only be accessed by one process at a time. The possible operations on the queue are to either put data on (send:!) or pull data off (receive:?). A channel's send and receive operations are random. The last sent item is not necessarily the next to be received when more than one element is on the channel. If a channel's buffer is full on a send operation the sending process blocks until the channel is no longer full. Similarly, if a channel is empty on a receive operation the channel will block until it is no longer empty.

Channels are initially specified with a buffer size that is a positive integer less than 256. A channel operates in one of two distinct modes which are defined by the buffer size. When the buffer size is zero the channel performs synchronous communication, otherwise the channel performs asynchronous communication.

A channel in synchronous mode forces a send operation in one process to be followed immediately with its respective receive operation in another process. This mode causes the sending process to surrender its execution to the receiving process, which is undesirable in modeling ad hoc protocols because it restricts the execution of communicating processes. For example, if node v_1, from Figure 2.1, sends a message, either v_0 or v_2 will immediately receive this transmission. If v_0 is the first to execute, it has the potential to deprive v_2 from sending any messages by sending a message immediately back to v_1 which will respond in kind preventing v_2 from every sending a message that is read by v_1.

Asynchronous communication is more flexible. In this mode, two processes can execute independently because the buffer size is at least one. Assume that each channel for the wireless medium is set to a buffer size of one. In Figure 2.1, if v_1 transmits a HELLO message, the two processes, v_0 and v_2, are free to either immediately read a message from the wireless medium or generate their own HELLO message. Both v_0 and v_2 are prevented from sending a message simultaneously because only one message can be held in the buffer going to the wireless medium. Neither process may permanently deprive the other of sending to the wireless medium.

The keywords *xr*, for exclusive receive, and *xs*, for exclusive send, are able to minimize the state associated with message passing. Both *xr* and *xs* aid in partial order reduction of the verifier because it allows the state associated with receiving and sending messages to be eliminated from specific channels.

Promela violates the traditional concept of loop and conditional structures. Unlike traditional procedural languages the *do* and *if* structures are best thought of as a C-style *switch* [1] except that each true case entry has equal probability of being executed and execution does not *fall through* to the next statement, see Table 2.7 for a comparison. Promela's behavior is abnormal because, traditionally, the first true case is always executed, whereas in Promela true case may execute.

Table 2.7: C *switch* statement [1] compared to a Promela *if* statement.

C *switch* statement (deterministic)	**Promela *if* statement (probabilistic)**

`switch (expression) {`
`  case const-expr: statements`
`  case const-expr: statements`
`  default: statements`
`}`
	`if`
	`::bool expression -> statements`
	`::bool expression -> statements`
	`::else -> statements`
	`fi;`

The rationale for Promela's probabilistic control structures is that distributed processes are not synchronized. A process has the ability to perform some other task while waiting for data. These same structures are not intended for deterministic control.

For example, when v_1, of Figure 2.1, sends its HELLO message the wireless medium process must send a message to every node connected to v_1. This operation requires iterating through the connectivity matrix to determine which processes must receive the message. It is possible to use Promela's *do* structure to achieve this behavior, but this adds unnecessary complexity to the model. A better solution is to define every possibility as a separate statement, which will always execute deterministically without abusing Promela's notation. A problem with approach this is that the such a model cannot scale to larger network sizes.

SPIN utilizes the standard C pre-processor by default . The C pre-processor, although powerful, lacks any functional features; it is typically employed to define key-value pair substitutions in a program. A more powerful pre-processor, *m4* [3], is available. The *m4* pre-processor provides greater flexibility with built-in support for recursive functions. Using recursion in *m4* it is possible to create scalable control structures and automate other solutions usually performed by external tools or scripts.

Using the features of SPIN, Ruys [33] lays the groundwork for the wireless medium in Promela. Originally the process was called a *broadcast server process*. Andel [2] further refines this method into a *wireless medium server*. At its core, the wireless medium server is the broker for all communications in the modeled wireless network. The server has the ability to enforce connectivity rules between network nodes. The rules are a set of links and can be defined in a Boolean matrix.

2.4 Chapter Summary

In this chapter the history of ad hoc networking was given. The classifications of ad hoc routing protocols were presented and several protocols, both secure and insecure, were introduced. Specifically OLSR and Secure OLSR were discussed in significant detail. The existing verification efforts for secure routing protocols were surveyed.

[3] http://www.gnu.org/software/m4

Finally, the pertinent details of SPIN and Promela for modeling ad hoc networks were expressed along with a brief background on modeling a wireless medium in Promela.

3 Methodology

Verifying the security properties of ad hoc routing protocols can be achieved with a combination of tasks. The process requires creative thinking and is by no means formulaic. One task is developing the protocol model. Formal properties must be specified as verifiable within the model. These verifiable properties may be useful in performing model validation. Validation of the model is critical to show that it performs in accordance with the protocol's specifications and cryptographic assumptions. Another task is to design an attacker with capabilities that may violate the protocol's security properties without violating the cryptographic assumptions. Attack models must be verified in all potential message exchange and attack interleavings. Following completion of the necessary tasks, the verifier is run given the protocol model, its security properties, and the attacker. This chapter develops these tasks into a framework using the Simple Promela Interpreter (SPIN) model checker. The framework is then applied to OLSR and Secure OLSR protocols.

3.1 Goals

The goal of this research is to propose and evaluate a framework for the automatic verification of proactive, table-based MANET routing protocols. The framework will determine if a protocol meets its intended security goals. If it does, then the framework generates a specific counter-example. The absence of a failure does not indicate that a protocol is secure. In building a verifiable routing protocol model there are three specific principles which drive the design of the final model:

1. Verification of routing security
2. Minimization of state-space associated with verification
3. Obey the protocol specification

Applying these principles drives the modeling effort toward the goal of creating a model which is both tractable, in terms of verification, and outputs useful information about the modeled protocol. Attacks against ad hoc routing protocols are often subtle. This subtlety can be directly related to apparent non-deterministic behavior. An appropriate ad hoc routing model must exhibit the same non-deterministic properties as a physical implementation of the protocol. For this reason it is critical to follow the protocol specification as closely as the modeling environment will allow.

3.2 Assumptions

Modeling secure ad hoc routing protocols requires several assumptions to be made. Under these assumptions the formal verification of each protocol can be performed clearly and concisely. Throughout Chapters 3 and 4 the following are assumed: bi-directional links, no mobility, no simultaneous transmissions, attacks only add false routes, cryptography is perfect, and benign nodes have a single identity.

Bi-directional links are assumed to exist between all networked nodes. Under this assumption a Boolean matrix representing connectivity is always symmetric. In the physical world bi-directional links are not always formed between nodes. Protocols must be able to handle cases where a node can receive information but is unable to send information, or vice versus. Handling non-bi-directional links has the potential to introduce additional security flaws. For this research only bi-directional link cases are examined, simplifying each protocol and reducing the set of possible network configurations.

No mobility of nodes is considered during execution of the models. By their specifications the models have the ability to handle node mobility; however, exercising this aspect of each model adds a considerable number of states to the overall verification. If modeling of mobility is desired for these models, then its implementation must occur in the model of the wireless medium. By assuming no mobility it is unnecessary to model link-breakage considerations that may be part of a protocol specification. The cases analyzed in this research consist of static link configurations between nodes, known as *topologies*. A security flaw one topology may not be possible in another topology. This static analysis may prevent potential attacks from being discovered, but has been applied by several other researchers [19, 2].

It is assumed that simultaneous transmissions are handled below the network layer. Specifically the Institute of Electrical and Electronics Engineers (IEEE) 802.11-2007 standard handles simultaneous transmissions with Carrier Sense Multiple Access/Collision Avoidance (CSMA/CA) at the data-link layer. The links between each node are handled at the physical layer. A model, called the *wireless medium server*, creates an abstraction for both the physical and the data-link layers. All node communication is arbitrated by the wireless medium server. The wireless medium server and protocol models work together to prevent simultaneous packet transmissions.

Only attacks against routing protocols that add false routing information are considered. All other attacks against the routing protocol fall under the category of denial of service. This assumption simplifies the definition of security by limiting its scope to finding the class of attacks that add inaccurate routes to a network. This assumption only affects the design of the security properties and is highlighted in Section 3.3.1.

Cryptography, alone, comprises an entire body of research. It is assumed that the standard cryptographic mechanisms used by routing protocols, mainly public key cryptography, are perfect. In addition, private keys are known by exactly one entity (node) unless otherwise noted, and all public keys are accurately known by all entities (nodes). The research goal is not to find flaws in the cryptographic primitives used by routing protocols, rather it is to find flaws in the ways that these primitives are used.

The final assumption is that every benign node in an ad hoc network maintains only one identity. This identity is synonymous with a network address and interface. Most routing protocol specifications allow for multiple interfaces per node, but this information is only used for making detailed forwarding decisions. Malicious node models are not limited to a single identity, they have the potential to use multiple identities and share their own cryptographic keys between any other malicious nodes. This capability allows malicious nodes to work together using pre-shared information in their attempts to break ad hoc routing protocols.

3.3 Framework

The goal of this research motivates a precise definition of routing security. Given the definition of security a method is required for formally checking if this definition is satisfied by a routing protocol. Armed with the proper definition of security and a verification method a framework is designed which automatically performs security verification on a specific routing protocol model.

The definition of security helps define the strategy for creating two security metrics, ϕ_{subset} and $\phi_{complete}$. The metrics, also known as properties, are not equally applicable to any protocol. Both assume that the underlying protocols distribute routing information between nodes. The property, ϕ_{subset}, is designed to work for any table-based protocol. The other property, $\phi_{complete}$, is limited to OLSR-like protocols.

Finally, a framework is presented which accepts a protocol specification and security property. The underlying verification tool is SPIN. Using the considerations for modeling MANETs in SPIN (see Section 2.3.3), the formal security verification framework's design is presented.

3.3.1 Security Metric. As defined by Andel [2], a routing protocol is secure if the routes it creates are accurate and these routes are always accurate. In order to evaluate the security of the protocol, a set of metrics must be implemented based on this formal security definition.

In Andel [2], the verification of accumulation-based, reactive routing protocols is straightforward. Every time a route is received its existence is verified with respect to the physical network. This check is straightforward because every verifiable route is explicitly returned in a RREP. Reactive protocols are also simplified because the routing process terminates. In proactive protocols the routing process exhibits infinite execution. Proactive protocols also require tables for storing hop-by-hop routing information. Both the infinite execution and the table-based characteristics make proactive protocols particularly tricky to verify.

In accumulation-based protocols a rule (assertion) can be placed in the requester node that verifies all returned routes are physically possible. The verification of table-based routing protocols is more complex. Table-based routing implicitly stores route information at every node in the network; therefore, a rule must be executed every time a node's routing tables are changed.

Proving correct operation, in proactive or table-based protocols, requires showing that routes are always accurate from the perspective of every node in the network. Reactive, accumulation protocols have a specific start and end to route discovery. The start is the initial RREQ and the end occurs when all RREPs have been received by the initiating node. For reactive protocols, it is sufficient to verify a property against all received RREPs. For proactive protocols a verification must show that a property always holds. In table-based protocols routing information is distributed between every node in the network, each table is subject to change at any time; therefore, it is necessary to show that the individual routing tables are always accurate.

There are several potential strategies that can be used to prove that a routing protocol functions correctly:

1. Source-destination route verification

2. Message flooding verification

3. Verification that routing tables form a subset of the link topology

Strategy 1 entails assigning one node a source identity and another node a destination identity, then testing to see if the routes between the source and destination are accurate. This method was applied in Andel [2] to accumulation-based, reactive protocols. Unfortunately, the endpoints in table-based protocols do not know the route between a source and destination. The only way to detect these source-destination routes is to send a message over them. To apply this technique to table-based protocols it is then necessary to model a data packet, for modeling the data forwarding phase of the protocol. This addition adds unnecessary overhead to the verification by requiring a model of for data forwarding. Implementing strategy 1 adds unnecessary complexity to the verification of table-based protocols.

Strategy 2 identifies a source node which sends a packet throughout the network. If the message is received at every node in the network, it is possible to conclude that routes exist between every node in the network (especially under the assumption of bi-directional links). In this strategy the routes used for the packet are unknown, making it impossible to identify if the paths taken by the message are in fact legitimate paths in the network. Strategy 2 is unable to detect the introduction of inaccurate routing decisions. This strategy differs from strategy 1 because the flooded packet is common to the route discovery phase of all table-based protocols. Strategy 2 provides the important ability to detect when existing routes remain undetected by route discovery.

Strategy 3 relies on the idea that the links formed in a routing table must be a subset of the links available in the physical network. Throughout the process any time a table is modified it is checked for accuracy relative to the true set of available links. If the table state is inaccurate at any point during route discovery, then the protocol has accepted an inaccurate route, i.e., it is insecure.

Strategies 1 and 3 can detect the addition of false routes. Strategies 1 and 2 can both detect when a route is denied. Given these two facts, strategy 1 would to be the clear choice for verification. However, it is not possible to implement this strategy in table-based protocols because the actual path is not known at any one node. One feasible work around is to embed the route taken by a packet. This workaround adds the burden of modeling message packets. The task of verifying when a route is denied, as in strategy 2, is unnecessary according to the formal definition of routing security. Therefore, strategy 3 is accepted as the best solution for verifying the security of a routing protocol. The 3rd strategy applies equally to any proactive or table-based routing protocol. The property, ϕ_{subset} given by Equation 3.1, utilizes this strategy.

Let the symbol ϕ_i represent the property that route i is accurate. Given N is the set of nodes in a network, let

$$\phi_{subset} = \phi_i \forall i \in N \tag{3.1}$$

Equation 3.1 (ϕ_{subset}) implies that every route corresponds to some set of physical links in the network topology and precludes the addition of routes that do not exist in the network. An attack that injects non-existent routes or links is detectable by ϕ_{subset}.

A definition of ϕ_{subset} in terms of set notation is provided in Equation 3.2. The formulation can provide a metric for determining when fake links are accepted in the network. Let $i.A$ represent the set of links node i has accepted, and B represent the set of existing (true) links. If $i.A \supset B$, then at least some routes in $i.A$ are inaccurate. When $i.A \subseteq B$ all routes in $i.A$ must be accurate, hence:

$$\phi_{subset} = \begin{cases} \text{true} & \text{if } i.A \subseteq B \\ \text{false} & \text{if } i.A \supset B \end{cases} \tag{3.2}$$

An alternative notation for expressing 3.2 is to write $|i.A \setminus B| == 0$. The expression evaluates to true if and only if $i.A$ is an improper subset of B, otherwise the expression evaluates to false (i.e., $i.A$ is a superset of B). This notation is particularly useful because, when using bit-vectors, set subtraction ($\setminus$) can be implemented in constant time with only bitwise operations.

To form Andel's definition of security, stating that ϕ_{subset} must always hold is equivalent to requiring that an adversary cannot add a fake route at any time. The expression *always* is a temporal statement, because it describes the proposition ϕ_{subset} over a period of time, in this case *always*. *Linear temporal logic (LTL)* provides the ability to express such statements involving time. Equation 3.3 is written in LTL and reads *always* ϕ_{subset} *is true*. The statement is false if ϕ_{subset} ever evaluates to false.

$$\Box(\phi_{subset}) \tag{3.3}$$

The property ϕ_{subset} cannot detect all actions of an attacker. Notably, ϕ_{subset} remains true even when existing routes are eliminated. An attacker capable of blocking existing links from use is not detectable by this property. It is the responsibility of the protocol to work around unusable or non-existent links and deliver traffic if the available links allow. An adversary has the potential to deny physical links in many ways, but most of these lead to immediate identification of the adversary's location(s). A known location enables targeted kinetic actions against the enemy. Although ϕ_{subset} does not detect denial of service attacks it is powerful enough to discover inaccurate routes.

A more stringent property, $\phi_{complete}$, is used in the validation of the model, as discussed in Section 3.5.3. This property can determine when existing links remain undetected. Let the symbol $\phi_{complete}$ represent the property that all neighbors are discovered and all messages flooded during route discovery (TC messages) reach all nodes. Equation 3.4 can be used to show that OLSR builds an appropriate view of the network. Section 3.5.2 implements $\phi_{complete}$ for the OLSR protocol.

$$\Diamond\Box\phi_{complete} \tag{3.4}$$

Both $\phi_{complete}$ and ϕ_{subset} were initially designed with the purpose of verifying security; however, $\phi_{complete}$ does not accurately test security because it is unable to handle situations where an adversary forms the only physical link between a network and some benign node. This situation is depicted in Figure 3.1. The use of $\phi_{complete}$ produces false security violations when the malicious node, v_A, fails to participate correctly in topology control. The false security violations occur when a node is connected to the network only through the adversary, v_A. $\phi_{complete}$ detects that an MPR was not forwarded beyond the attacker. This is not a security violation because the adversary does not create an inaccurate route, yet $\phi_{complete}$ is violated in this situation.

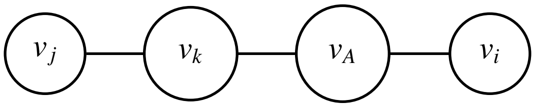


Figure 3.1: The malicious node, v_A, forms the only link between v_i and the nodes v_j and v_k.

The property, $\phi_{complete}$, is useful in validation *i* because it offers a more complete protocol check, capable of demonstrating that MPR selection is consistent throughout the network. As soon as an adversary is introduced $\phi_{complete}$ becomes useless.

Validation and security verification are both assisted by ϕ_{subset}. Thiiss property only tests to see that the set of MPRs learned at every node are actually possible in the network. The property follows directly from the definition of security that prohibits the use of inaccurate routes. Based on the definition of security, there are some results that can be readily predicted for an attacker which assist in validation of the model. Also, because

ϕ_{subset} matches the definition of security it is natural to use this property in security verification.

3.3.2 Selecting the Verification Tool. Of the security verification methods available, see Chapter 2, formal methods is the most appropriate to use with ad hoc networks. This follows because formal methods have the ability to detect subtle errors in a distributed system. There is a broad range of tools available for executing the formal methods approach.

The selected formal method must provide enough generality to capture the specification of any ad hoc routing protocol, be strict enough to avoid common errors, and be flexible enough to avoid unnecessary implementation level details. These three criteria preclude formal methods based in process calculus and strand-spaces because they are too abstract to avoid common errors. The immediate problem being that these two methods rely heavily on mathematical assumptions about the environment that cannot be tested. The types of formal methods we choose to examine are those that enforce a structured grammar. It is this limitation that allows the specification of a protocol to avoid common errors.

SPIN [31] is a tool based in formal methods that is designed to provide an exhaustive verification of distributed systems. Ad hoc networks are a form of a distributed system and some researchers have applied SPIN in proving limited properties about specific ad hoc routing protocols. Section 2.3.2 provides a review of research applying SPIN to ad hoc networking. Process calculi provide proof systems with the opposite goal of SPIN. SPIN has advantages over Buttyán and Thong's [28] process calculus:

- SPIN allows models near implementation level rather than requiring abstract mathematical models.

- SPIN's verification process is exhaustive, the use of Büchi automata aid in partial order reduction of the model's state. In process calculi the models properties cannot be partially proven.

- SPIN's modeling environment provides an intuitive relationship to the distributed process associated with ad hoc protocols. Again compare this to the abstract models required for process calculus.

Only AVISPA and SPIN can meet the research goal. SPIN's design abstraction, state reduction, and state-space compression techniques have made it possible to verify the properties of at least five node reactive, accumulation-based routing protocols. SPIN has several clear advantages over AVISPA:

- SPIN is very mature and has been in development since 1989 whereas AVISPA was developed in 2003.

- SPIN is designed with the goal of modeling and verifying distributed systems whereas AVISPA's core tools have been designed specifically for cryptographic analysis.

- SPIN is maintained as a single unified tool, versus AVISPA's use of the CL-AtSe, OFMC, and SATMC tools.

- SPIN has several state-space reduction techniques built-in which can be invoked when exhaustive verification fails. This process is in contrast to the *push-button* verification scheme boasted by the AVISPA project.

SPIN also provides functionality for LTL which is necessary for capturing the time-based variation of ϕ_{subset} and $\phi_{complete}$. In SPIN this functionality is achieved through the use of the *never claim* (see Section 2.3.3). The claim examines the state of every process in the model and determines if the property holds. By comparing each node's routing table to the true network topology it is easy to determine that no extra routes have been added to the network. The LTL provides the necessary utilities for verifying that the $\phi_{complete}$ and ϕ_{subset} properties hold throughout the operation of the protocol. In this way, a routing protocol model can be completely verified.

3.3.3 Design. The purpose of the framework is to verify security of a routing protocol, in other words, that nodes learn only accurate routes and that those routes are always accurate. This can be achieved by encoding a model in a context that supports the security verification. Three major inputs to the model are the network's connectivity graph, T, a specific attacker model, *A*, and the correctness property, ϕ. The output is *FAIL* if a security violation occurs for the set of inputs. An output of *PASS* implies only that the protocol is secure for the given set of inputs. With an output of *FAIL*, the verifier produces a *trail* file containing the states and transitions that cause the violation. Figure 3.2 depicts this framework given the model, *M*.

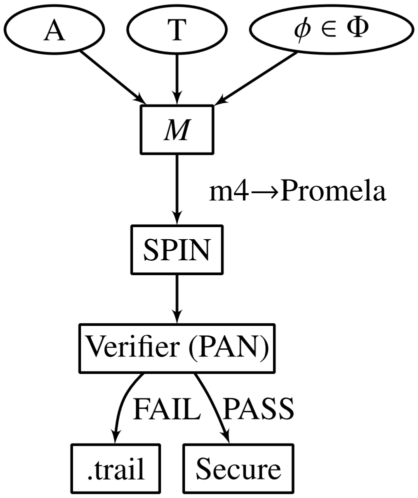


Figure 3.2: Framework for model verification.

Table 3.1: Bit String Format, N=4

Binary Digit	5	4	3	2	1	0
Edge	$v_2 - v_3$	$v_1 - v_3$	$v_0 - v_3$	$v_1 - v_2$	$v_0 - v_2$	$v_0 - v_1$

The two properties ϕ_{subset} and $\phi_{complete}$ both are elements of Φ. Other properties may be added to Φ for use in verification. The function $topo(T)$, see Listing 3.1, maps the input T to the appropriate network topology. The input A is manually mapped to a set of attack vectors.

The structure of T is adapted from [34], where a network topology is stored as an integer. Each binary digit of the integer represents an edge in the topology. This encoding is presented in Table 3.1. Each edge defines a pair of nodes, edge(v_i, v_j), which are either connected, 1, or disconnected, 0. This property is expressed in Equation 3.5. Each edge is bi-directional, as captured by Equation 3.6. No node is allowed to transmit to itself; therefore, whenever $i = j$ the two nodes are considered disconnected, this property is defined by Equation 3.7.

$$\text{edge}(v_i, v_j) \in \{0, 1\} \tag{3.5}$$

$$\text{edge}(v_i, v_j) \equiv \text{edge}(v_j, v_i) \tag{3.6}$$

$$\text{edge}(v_i, v_j) = 0 \text{ if } i = j \tag{3.7}$$

An edge is uniquely mapped to every binary digit of an integer T. The algorithm in Listing 3.1 provides one such mapping. The resulting array of edges has the properties expressed in Equations 3.5-3.7.

With Listing 3.1, it is possible to enumerate all potential networks for N nodes. To achieve this we start by finding the upper bounding topology, T^{Ω} for a network with N nodes. Given bi-directional links, there are $(N(N-1))/2$ potential edges. Edges are binary

Listing 3.1: Function *topo*() for mapping T to a network topology.

```
row=0; column=0; n=0;
topo(T) {
   if(t>0) {
      if(row ≥ column) {
         row = 0;
         column++;
      }                              //Eq 3.5: 0 ∈ {0,1} and T%2 → {0,1}
      edge[n,n] = 0;                 //Eq 3.7
      edge[row,column] = T%2;        //Eq 3.6
      edge[column,row] = T%2;        //Eq 3.6
      row++;
      n++;
      topo(T/2);
   }
   return edges;
}
```

(Equation 3.5), so there are $2^{(N(N-1))/2}$ unique ways to combine the edges. Finally T^{Ω} is calculated with Equation 3.8. [4]

$$T^{\Omega} = 2^{(N(N-1))/2} - 1 \tag{3.8}$$

For the purposes of this research, the number of nodes, N, is bounded. Andel [2] limited reactive protocol verification to five-node networks. The model described here is designed to scale up to eight nodes; however, N was limited to five nodes for analysis. Exhaustive verification has been applied to four nodes, and non-exhaustive verification has been applied to five nodes. In the OLSR model, four nodes reaches the practical limits of available system memory (32 gigabytes) for exhaustive verification. When $N = 4$ there are 64 potential topologies and when $N = 5$ there are 1,024 potential topologies. Not all topologies need to be analyzed due to symmetric properties exhibited by some networks, see [34] for an explanation. Furthermore, the topologies in an N-node network is a subset

[4] T^{Ω} is 1 less than $2^{(N(N-1))/2}$ because the lower bounding topology is 0.

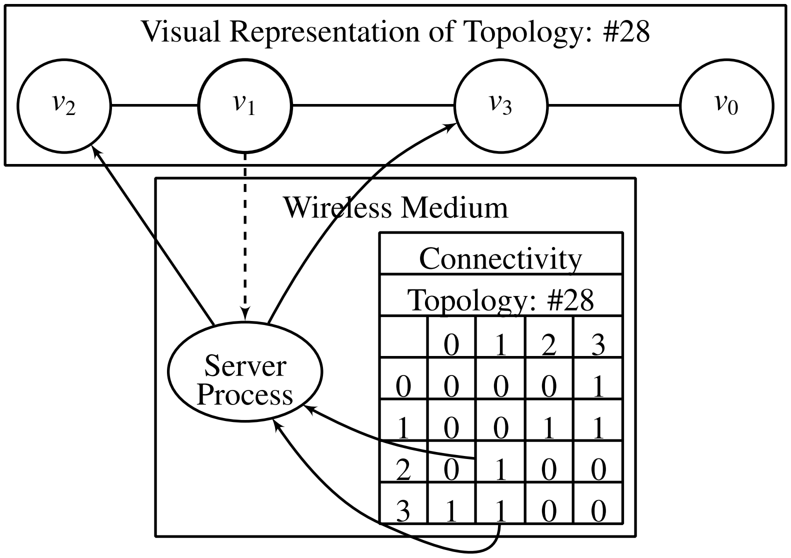
of those in an (N + 1)-node network; therefore, verification of all (N + 1)-node networks implies verification of all N-node networks.

For a large N, the verifier may never be able to explore all the states in all topologies. Limiting verification to an estimate of exhaustive verification greatly reduces the time and memory required to analyze a protocol. Most property violations should be revealed after exploring only a few possible combinations of states. Upon reaching the first violation, the protocol is deemed insecure. Security cannot be inferred when the estimated verification completes because all possible combinations are not explored.

Modeling of the wireless medium in Promela requires a separate process to handle broadcast communications. This process is used to enforce the physical restrictions of a wireless topology. Even adversarial nodes are restricted to the topology rules of the wireless medium. See Section 2.3.3 for the origins of the wireless medium server in SPIN.

The wireless medium server used in this research is depicted in Figure 3.3 and the associated Promela pseudo-code appears in Listing 3.2. One key difference from Andel's server is that the connectivity matrix, *con*[id], is expressed as an array of bit-vectors (bytes), rather than a two-dimensional array of bytes. This change alone reduces the wireless medium's state-space requirement from O(N^2) to O(N). Another change is to set the buffer size of channels *to_wm*, to wireless medium, and *from_wm*, from wireless medium, to one. Every node must check that the *to_wm* buffer is empty before sending. This check is performed as an atomic operation. As a result, a node never blocks while waiting for a buffer to clear. This restriction allows the sending node to perform some other action until the *to_wm* channel is clear. This approach further decreases state-space because the channel buffers are as small as possible, size one, without triggering the synchronous channel mode.

The *m4* pre-processor (see Section 2.3.3) is used to improve the scalability of the model. Specifically the code for the wireless medium server and associated connectivity



Connectivity Topology: #28				
	0	1	2	3
0	0	0	0	1
1	0	0	1	1
2	0	1	0	0
3	1	1	0	0

Figure 3.3: Node v_1's communication is controlled by the wireless medium.

matrix is created dynamically. These two segments of code depend on the number of nodes which the *m4* macro infers from the topology flag **T**. Also, the definitions of attackers are referenced at compile time. The flag *A* represents a specific attack model which is defined in an associated file, *attacks.m4*. This file separates the attacker code from the correctly specified protocol code. Various parameters and capabilities associated with an *attack vector* are coded into the file. The use of attack vectors simplifies the analysis of security by explicitly grouping a set of attacker capabilities and its associated number of attacking nodes as a single entry.

In order to limit the state-space for of the model several decisions are made. Throughout the model most global variables are hidden from the verifier because they do not reflect routing state for any specific node. These variables are declared using the *hidden* keyword. In the wireless medium, nodes exclusively send (*xs*) to the medium on its input channel. Also, the medium exclusively sends on its outgoing channels (one per node), which translates to an exclusive receive (*xr*) for the respective receiving node. Each

Listing 3.2: Wireless Medium Server Process.

```
chan to_wm =   [1] of type {byte, byte...}
chan from_wm = [1] of type {byte...}
hidden byte data...         // Declare all data fields
hidden byte id;             // Declare id field

proctype Wireless_Medium()
{
  xr to_wm;                 // Set to_wm to exclusive receive

  for(i : nodes)            //m4: Rpt nxt 2 statements for all i
    from_wm_i = from_wm[i]; // Copy from_wm[i] to from_wm_i
    xs from_wm_i;           // Set from_wm_i exclusive send

  do                        // Run server process indefinitely
  ::to_wm?id, data... ->    // Listen form messages
    for(i : nodes)          //m4: print if statement for all i
      if                    // If id--i (link) exists,
      ::IS_1(con[id],i) ->  //   test con[id] for node i (macro)
        from_wm_i!data...;  //   then send data... to i, proceed
      ::else -> skip;       // else next statement
      fi;
  od;                       // Rinse and repeat
}
```

node also maintains a set of scratch variables used for temporarily storing received data. The scratch variables may be used for other purposes, but after use they are always restored to a null state. The verifier does not have to track the status of a null variable, thus reducing the state-space. The *m4* pre-processor is useful in creating aliases for scratch variables that reflect the specification's terminology.

One of the greatest challenges in modeling routing protocols in general is using efficient data structures. Promela is not a general purpose language so it lacks many of the libraries that would be considered elementary for most programming languages. Routing tables can be represented as bit vectors. Each binary digit's position is reserved to represent a node's network address. The least significant bit corresponds to node v_0 and the n^{th} bit corresponds to node v_{n-1}. The basic bit-vector operations are defined using Promela's bitwise operators and *m4* macros are used to improve this code's readability. An implementation of the *Hamming Weight* function is employed to calculate the number of elements stored in a bit-vector (number of bits set) in a constant number of operations. Also, the bit-vector data-structure allows the modification and reading of data to be performed in O(1) operations. The number of bits required to represent all nodes is equivalent to the number of nodes in the network, thus a byte (eight bits) can represent eight nodes. Bit-vectors completely eliminate the need to iterate over elements for any operations, significantly reducing complexity and overhead.

Promela's *remoteref* mechanism is used to allow a *never* claim to access each node's state without modifying it. The use of *remoterefs* comes at the cost of abandoning partial order reduction. A preferred solution, restoring partial order reduction, would be to declare each node's state globally with the *local* keyword. Despite this *remoterefs* are used in the modeling effort.

The file, *attack.m4*, provides a one-to-one mapping between an integer, $A > 1$, and an attack vector. $A = 1$ is reserved for the benign operation of a protocol, i.e., no attackers are present. An attack vector is defined by a Promela *proctype* and number of attackers. The *proctype* provides a definition of the attacker and its associated capabilities. In general, the attacker is a specialized node designed to systematically violate assumptions of the current routing protocol. Some attacker definitions are capable of collusion, meaning they can work with other nodes in an attempt to violate security properties. An attack vector can define one or more malicious nodes. Depending on the definition of an attacker it may or may not collude with other attackers.

3.4 Attack Vectors

An attack vector is defined as a tuple containing the number of attackers, a trust relationship set, and an attacker definition. An attack vector, A is characterized by the

Table 3.2: Trust Relationships ($\mathbb{T}$) in Secure OLSR. **B** = {all node identities}

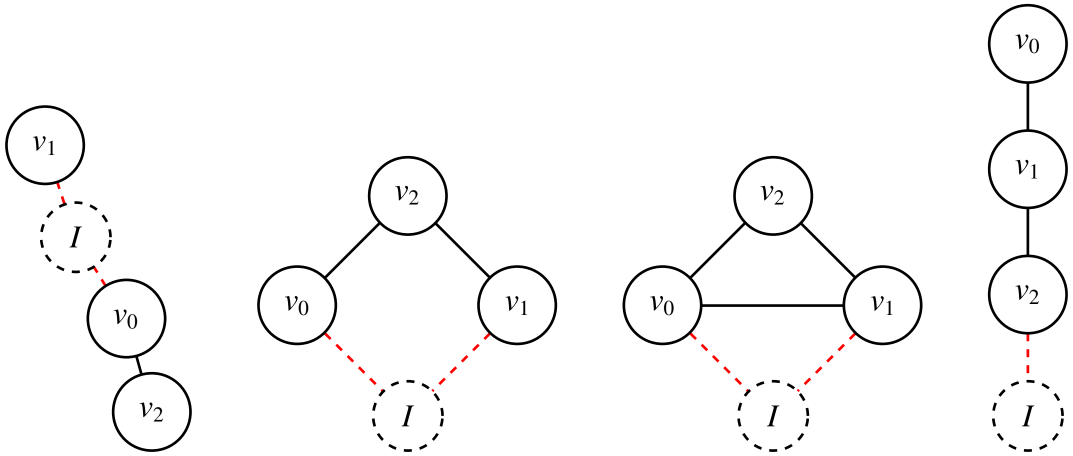
$\mathbb{T}$	$\mathbb{C}$	$\mathbb{U}$	Description
B	{}	{}	Only trusted identities are present.
B	$\subseteq \mathbb{T}$	{}	One or more compromised identities.
$\mathbf{B} \setminus \mathbb{U}$	{}	$\mathbf{B} \setminus \mathbb{T}$	Both trusted and untrusted identities.
$\mathbf{B} \setminus \mathbb{U}$	$\subseteq \mathbb{T}$	$\mathbf{B} \setminus \mathbb{T}$	One or more compromised and untrusted identities.

tuple $A = (N_a, \mathcal{T}, \mathcal{D})$ where N_a is the number of attackers, $\mathcal{T}$ is the set of trust relationships, and $\mathcal{D}$ is the definition of the attacker.

The attacker definition, $\mathcal{D}$, is limited to providing the same capabilities that are available to benign nodes. The definition cannot provide an attacker with the ability to break cryptography based on the assumptions stated in Section 3.2. An attacker must obey the rules and connectivity of the wireless medium. These limitations do not preclude a slew of other possible behaviors. For example, two malicious nodes may be designed to collude with one another and both node's public and private keys may be shared. These undefined behaviors allow malicious nodes to form complex attacks against a protocol.

The trust relationship set, $\mathcal{T}$, describes the state of every identity's public and private keys in the model. A node's identity is an element in the trusted set, $\mathbb{T}$, if all benign nodes possess the node's associated public key. A node's identity is in the compromised set, $\mathbb{C}$, if an attacker knows the private key for the node's identity. A node's identity is in the untrusted set, $\mathbb{U}$, if it is not in the trusted set, $\mathbb{T}$. The tuple $(\mathbb{T}, \mathbb{C}, \mathbb{U})$ defines a trust relationship in Secure OLSR. All benign node identities are in $\mathbb{T}$. The benign nodes do not know which identities are in $\mathbb{C}$. The possible combinations for the tuple $\mathcal{T}$ are presented in Table 3.2. The case where $\mathbb{T} = \{\}$ is not considered because it is assumed that some set of benign nodes exists and that these nodes have all established trust with one another.

An attacker's actions are tested against the protocol. Depending on the actions available a protocol may lead to the discovery of an unknown attack. It is also possible to create an attacker such that it recreates a previously known attack sequence. Take, for instance, the *Invisible Node* attack [35]. This attack consists of a malicious node that relays data without modifying it. Due to this behavior otherwise disconnected nodes will appear to be connected, but the malicious node can, at any point, filter non-control data to make the link useless. This attack is effective in two cases. The first is if a benign node's sole connection to the network is through an invisible node, as shown in Figure 3.4a. The second is when an invisible node adds an otherwise non-existent link to the network, as depicted by Figure 3.4b. Cases where the invisible node is unable to add inaccurate routes are presented in Figure 3.4c and 3.4d.



(a) FAIL

(b) FAIL

(c) PASS

(d) PASS

Figure 3.4: Collection of topologies involving an invisible node, *I*. *FAIL* implies *I* adds at least one false route. *PASS* implies *I* has no effect.

Five attack vectors are formulated in this section. Three of these attacks can be considered *toy* attacks because they would be impractical against a network in which no

Table 3.3: Attacks Against Secure OLSR. Attack Vector: $(N_a, \mathcal{T}, \mathcal{D})$: N_a is number of attackers, $\mathcal{T}$ is a trust relationship, $\mathcal{D}$ is the definition of the attacker's capabilities.

A	Attack Vector	$\mathcal{D}$: Attacker Definition
i	$(0, (\{\mathbf{B}\}, \{\}, \{\}), \mathcal{D})$	An attacker is not defined, all nodes are benign.
ii	$(1, (\{\mathbf{B}\}, \{v_{N-1}\}, \{\}), \mathcal{D})$	Byzantine node, v_{N-1}, false routes added to own TC messages.
iii	$(1, (\{\mathbf{B}\}, \{\mathbf{B}\}, \{\}), \mathcal{D})$	Modifies received TC messages, all id's compromised.
iv	$(1, (\{\mathbf{B}\}, \{\}, \{\}), \mathcal{D})$	Relay received traffic, "Invisible node attack."
v	$(1, (\{\mathbf{B}\}, \{\}, \{v_{N-1}\}), \mathcal{D})$	Modifies received TC messages, no id's compromised.

private keys are compromised. Despite this, these attacks aid in the validation of the secure protocol model. Two other attacks are more realistic in that they do not require the use of any compromised cryptographic material. All five attacks have outcomes which can easily be derived without the verification framework; therefore, the expected results can be contrasted with the achieved results to demonstrate that the model does the right thing. Table 3.3 provides a summary of the five attack vectors. A roman numeral is assigned to each attack vector for reference.

Attack *i*: Attack *i* is the benign case where no attacker is imposed on the network. The vector $A = (0, (\{\mathbf{B}\}, \{\}, \{\}), \mathcal{D})$ describes a case where there is no attacker, and the trust relationship consists solely of trusted, uncompromised identities. This case is of interest because it shows that the secure protocol functions correctly under normal circumstances. In fact, Secure OLSR functions no differently than OLSR under attack vector *i*.

Attack *ii*: Attack *ii* is an attack in which a trusted node, containing its own key, falsely advertises routes in the network. The attack has the effect of adding false advertisements from itself to any other nodes. This vector is $A = (1, (\{\mathbf{B}\}, \{v_{N-1}\}, \{\}), \mathcal{D})$, thus one malicious node is defined and it assumes the identity of node v_{N-1}. This attack requires knowledge of a single trusted node's private key. This type of attack is commonly referred to as a Byzantine, or trusted insider, attack because the trusted node deviates from standard operating procedures in a potentially threatening way.

Attack *iii*: The vector $A = (1, (\{\mathbf{B}\}, \{\mathbf{B}\}, \{\}), \mathcal{D})$ details an attack where all trusted identities are compromised. One attacker is defined that has access to all compromised private keys. This scenario gives the attacker the ability to change every control message. The attacker definition, $\mathcal{D}$, limits itself to modifying TC messages which it is assigned to forward or create by the protocol. This attack describes an attacker that is both Byzantine and maintains multiple identities, commonly referred to as a Sybil attacker. Since the safeguard provided by private keys is removed a secure protocol cannot counter such an attack. This type of attack is extremely difficult to produce in practice because private keys are generally guarded with the utmost care. This attack is explored because of its ability to demonstrate the correct function of a secure protocol.

Attack *iv*: The invisible node attack is a simple, robust attack. Its associated vector, $A = (1, (\{\mathbf{B}\}, \{\}, \{\}), \mathcal{D})$, features the same trust relationship as vector *i*; however, one attacker is defined that relays control traffic to its neighbors without modification. Relaying can add links without knowing any cryptographic material. This attack is an example of an outsider with the ability to add false routes to the network. Countermeasures to the Invisible node attack rely on precise positioning and timing information, one such scheme is described in [36].

Attack *v*: In attack *v*, $A = (1, (\{\mathbf{B}\}, \{\}, \{v_{N-1}\}), \mathcal{D})$, the untrusted node, v_{N-1}, modifies and forwards TC messages; however, the node is unable to generate a valid signature for the new messages because it does not know any trusted private keys. All messages modified by node v_{N-1} have a non-valid signature; therefore, all modified control messages are ignored. Vector *v* is an example of an ineffective outsider attack against

Secure OLSR. The attack is useful in demonstrating that the model does the right thing by preventing such a simple attack.

The attacks discussed in Section 3.4 are specific to the OLSR and Secure OLSR protocols. Other table-based protocols do not necessarily share the same features. Attack vectors must be redefined for each specific protocol.

3.5 OLSR Model

Previous verification efforts have only analyzed reactive routing protocols. Verifying reactive protocols is not trivial; however, the task is certainly less complex than verifying a proactive protocol. OLSR is a proactive, table-based MANET routing protocol. OLSR does not provide any security; therefore, the protocol should be vulnerable to any simple attack that adds a malicious route to the network. Complete specification of OLSR is available in [16], see Section 2.1.3.3 for an analysis of OLSR. The model of OLSR presented here will serve as a baseline for demonstrating that proactive and table-based protocols can be modeled for formal security verification.

3.5.1 Model. Proactive protocols periodically send updates and these updates are sent indefinitely. Proactive protocols are said to exhibit infinite execution. In the case of OLSR, HELLO messages are generated indefinitely. Currently the design of the wireless medium server makes it impossible to model the infinite execution of OLSR. Infinite execution is prevented in the model by eventually stopping updates. This behavior is achieved by reaching a *steady state*. For OLSR, a steady state is defined as the point at which every node's Neighbor and MPR Selection sets no longer change. Achieving this state is accomplished by restricting when a HELLO message can be sent. Steady state is possible only under the assumption of no node mobility and it maintains the semantics of the protocol for the purpose of modeling.

In order to achieve steady state, the model is limited such that only one node is initially allowed to send a HELLO message. The identity of the initiating node is not important; however, it must be connected to the network topology. If it is not connected, then the HELLO is never received by any other node, leading to premature steady state. In the model, v_0 is always selected as the HELLO initiator. [5] Upon processing a HELLO message a local Boolean flag, say c, is set to true if either the Neighbor or MPR Selection sets change. The node may transmit a HELLO message as long as c is true. The variable c is set to false when a HELLO message is sent by the node. With these minor modifications, the infinite execution of OLSR is mitigated and the non-deterministic interleaving of messages is maintained.

In the case of the MPR Selection algorithm it is easy to assume that the selected nodes will always be sorted in the same order. For example, if two nodes have the same degree then the node with the lower network address is always selected. This last selection decision is not defined in RFC 3626; therefore, the result of the sort operations is not necessarily deterministic. To reflect non-determinism, the sort function initially chooses either the lowest or highest network address when there is a tie. Unfortunately, if there is more than a two-way tie, no nodes between the highest and lowest values will be selected as an MPR. Future versions of the model should be modified to reflect such possibilities.

In the OLSR model, the MPR Selection algorithm alone would make any verification impractical; however, the use of the deterministic code block, *d_step*, removes visibility of all but the final results of MPR Selection. Similarly, *d_step* can be applied to new and forwarded TC messages. By delaying their transmission and maintaining a forwarding list of the TC messages the internal transitions associated with message generation and processing is hidden. Data are stored in each node's topology control table. TC messages in the forwarding list must enter contention for the node's transmitter and the wireless

[5] The physical network topologies are limited to only those topologies in which v_0 is connected. This decision reflects that each node is homogeneous and prevents a premature steady state.

medium. The forwarding list slightly increases the state required per node but significantly reduces the number of reachable states by hiding the transitions associated with generating and forwarding TC messages.

The model of messages are composed as described in Tables 2.3 and 2.4. Table 2.2 reflects the local state variables in the OLSR model. The state-diagram, Figure 3.5, is generated directly from a Promela model of OLSR with three nodes. The state-diagram captures all finite automata states visible to the verifier for each node. On receive (< TC > or < HELLO >) and transmit (TC or HELLO) transitions the model is either receiving or sending a message from the wireless medium process. A detailed discussion of the wireless medium process is given in Section 3.3.3.

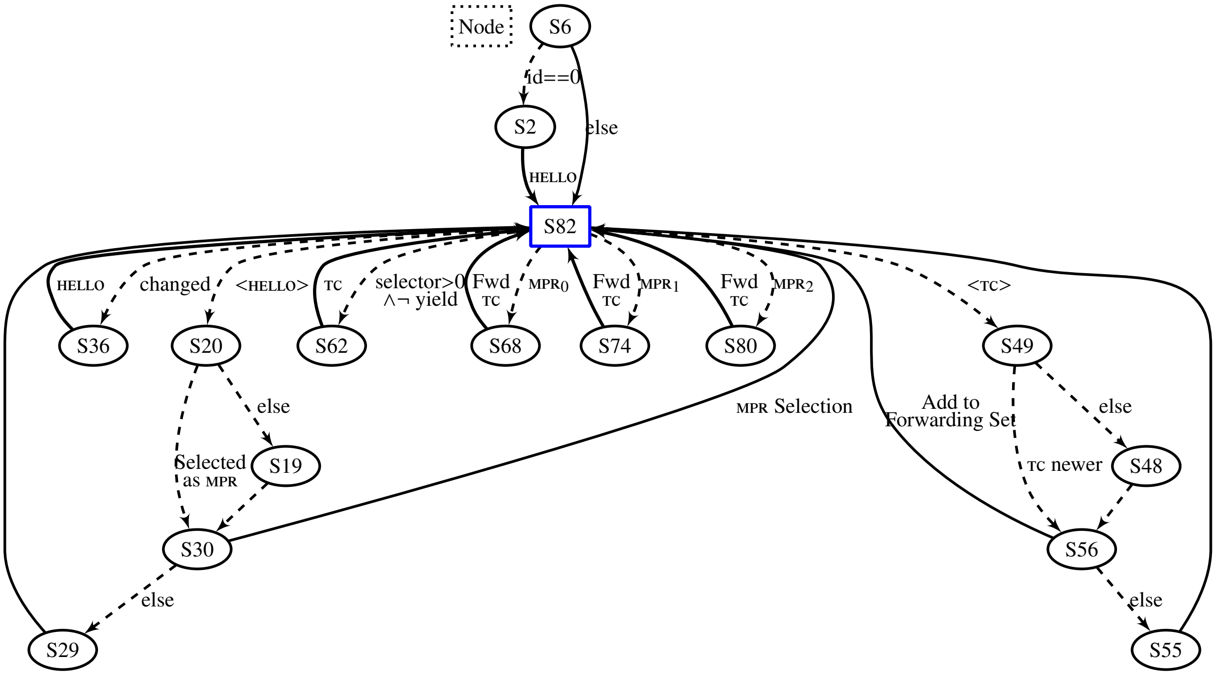


Figure 3.5: State machine for OLSR. State numbers are assigned automatically by SPIN.

3.5.2 Model Validation. Validation is a necessary goal for ensuring that each model accurately reflects the intended protocol. Four resources for meeting this goal are the protocol specification, SPIN in simulation mode, SPIN generated finite automata diagrams, and SPIN in verification mode.

Each Promela model is based on an author's description of the protocol. The modeler makes decisions based on the protocol's specification. This leads to a basic, intuitive validation of the model. This form of validation alone is insufficient because the protocol can be misinterpreted.

The Promela model can be run in SPIN's simulation mode to strengthen the intuitive validation. When a model is run in simulation mode a single, random seed is selected which determines how the model's communicating processes will be interleaved. During a simulation the user can select to output the messages exchanged in the system, the complete state of each process, or any other print statements as needed. A combination of these outputs can show how the protocol performs with the current seed. A logical comparison of the protocol's outputs against the prior expectations validates the model. A few random simulations, monitoring this invariant, provide confidence that the invariant is not violated. However, it is important to note that random simulation is insufficient because it is possible for a different seed to produce a case where the observed behavior does not hold.

Another, more practical, tool available from SPIN is a diagram of a model's finite state automata. Given the finite automata it is possible to map every model decision to one that would be made by the protocol specification. Figure 3.5 provides an example for the model of OLSR. Every state transition relates to an action in the OLSR specification.

The strongest tool available for validation of the model is SPIN in verification mode. Verification allows for the testing of an invariant, such as *only nodes with a non-empty selector set generate topology control messages*. Completely validating the model in this manner would be prohibitive because there are too many trivial properties. However, it is possible to use a property, for example $\phi_{complete}$, to validate that the larger goals of a protocol specification are met under normal operating conditions. In this validation process $\phi_{complete}$ is being used to show that route discovery messages reach every node and that only accurate routes are created, as discussed in Section 3.3.1. For this step the consideration of the benign attack vector, $A = 1$, is sufficient to provide validation.

The property, $\phi_{complete}$, needs to be implemented for the OLSR protocol. There are two distributed views of the network that can be used for implementing $\phi_{complete}$. These are the views provided by each node's Neighbor and Topology Control sets. Both views combined form a subset of the existing link topology.[6] The verifiable property becomes that every node: learns all of its neighbors, learns every topology control decision (MPR set), and that all of the links in these views are accurate.

In Equation 3.9, V is the set of all nodes, $i.\textsc{n}$ is the set of nodes neighboring i, and $con[i].\textsc{n}$ represents the set of nodes with edges to node i. This property expresses that all discovered neighbors are in fact neighbors in the real topology. If they are not, then the node's neighbor set must be different from the connectivity matrix, *con[i]*. This property initially evaluates to false but should eventually, always become true.

$$A = \forall i \in V(i.\textsc{n} = con[i].\textsc{n}) \tag{3.9}$$

In Equation 3.10, V is the set of all nodes, $i.\textsc{mpr}[j]$ is the set of nodes selected by j as MPRs for j, and $con[j]$ represents the set of nodes with edges to node j. This property captures the behavior that a node's MPR set is shared between all nodes in the network. If no MPR sets are formed, then this property is true. If node j has no neighbors, then $\neg con[j]$ evaluates to true and j's topology control status is not considered. Initially, property 3.10 will evaluate to true because all MPR and Topology Control sets are initially empty. Once one or more MPR sets are formed the property will evaluate to false. After

[6] The network view is a subset because the use of MPRs masks some potential links.

these changes propagate through the network, property 3.10 should eventually, always evaluate to true.

$$B = \forall i \in V(\forall j \in V(j.\textsc{mpr}[j] = i.\textsc{mpr}[j] \lor \neg con[j])) \tag{3.10}$$

The propositions 3.9 and 3.10, can be combined, leading to the final proposition $\phi_{complete} = A \land B$. This implementation captures the behavior that TC messaging disseminates the MPR selection information to all nodes in the network i.e., Strategy 2 in Section 3.3.1. If the MPR sets never provide complete network coverage, then $\phi_{complete}$ evaluates to false. Using this implementation of $\phi_{complete}$ it is possible to validate OLSR's neighbor sensing and topology control. The LTL formula of Equation 3.4 is used to verify that $\phi_{complete}$ is eventually always true. If this claim is not accurate in all benign cases, then the protocol model does meet the protocol specifications.

3.5.3 Security Verification. The OLSR protocol is not specified with the goal of security and it has no security features. The trust relationship, $\mathcal{T}$, associated with the attack vectors is meaningless, OLSR accepts all nodes as trusted. This blind trust means that an adversary can easily add false routes to the network.

Using the security property, ϕ_{subset}, and the topologies $\mathbf{T} = \{0..63\}$, it should be possible to show that OLSR is vulnerable. First attack vector i is tested. For this experiment it is expected that the framework outputs PASS in all cases because no malicious nodes are used. Next attack vector v is tested. Under this attack the malicious node v_{N-1} is expected to easily add false routes, resulting in the output of FAIL in any configuration where v_{N-1} is selected as an MPR. The results of these two experiments affirm the expectations, as shown in Chapter 4.

3.6 Secure OLSR Model

Secure OLSR is a minimally modified version of OLSR that adds security. The modifications to OLSR do not change its fundamental operation. Because Secure OLSR closely resembles the original most of the model features still apply. This version of OLSR adds two functions *sign()* and *verif()* which together form the basis for securing its operation. Section 2.2.3 provides the details on how these two functions provide security to OLSR.

3.6.1 Model. A model of Secure OLSR must provide implementations for *sign()* and *verif()*, Equations 2.1 and 2.2 respectively. The implementation forgoes cryptographic operations necessary in a real-world implementation as long as all nodes, including adversaries, are bound to the rules of a trust relationship, $\mathcal{T}$. The trust relationship, $\mathcal{T}$, is defined in Section 3.4.

The ideal implementation of *sign()* maps a unique value to every {private key, message} pair. The only way to generate the unique signature is to possess the private key. Let us define $\mathbb{T}$ as the set of trusted node identities, and $\mathbb{C}$ as the set of trusted but compromised node identities, and $\mathbb{U}$ as the set of all untrusted node identities. The signing function can then be expressed in Equation 3.11, where *id* replaces both the private key (k_{id}^{-}) and *message*, and *identity* represents the true identity of a benign node.

$$sign(\text{id } a, \text{message } b) \rightarrow \begin{cases} a & \text{if } id == identity \\ a & \text{if } id \in \mathbb{C} \\ j, j \in \mathbb{U} & \text{otherwise} \end{cases} \tag{3.11}$$

The implementation of *sign()* in Equation 3.11 ignores the *message* field which means that there is no way to associate the original message with the result of the function. The solution is to always send the result of *sign()* along with its corresponding message. This solution is allowed according to the Secure OLSR specification.

When a control message, say $< origin, message, signature >$, is received it is verified by *verif()*. If the function call returns “valid” then $message$ is processed in accordance with OLSR, otherwise the control message is discarded. It is assumed that the message corresponding to the signature is received in the same packet. Equation 3.12 provides a model of *verif()*:

$$verif(\text{id } a, \text{message } b, \text{signature } c) \rightarrow \begin{cases} `valid' & \text{if } a == c \wedge a \in \mathbb{T} \\ `valid' & \text{if } a == c \wedge a \in \mathbb{C} \\ `invalid' & \text{otherwise} \end{cases} \tag{3.12}$$

The first case in Equation 3.12 returns *valid* when a message is received from a trusted node. The second case in Equation 3.12 returns *valid* if a message is received that is using a compromised node identity. In all other cases the function will return *invalid*.

The modeled implementations of *sign()* and *verif()* are added to the OLSR model. To implement Secure OLSR: Every HELLO and TC message is sent along with the result of *sign()*, Equation 3.11. Every HELLO and TC message received is only processed if *verif()*, Equation 3.12 returns the string *valid*.

3.6.2 Model Validation. Validation of Secure OLSR requires showing that the *sign()* and *verif()* functions are implemented correctly. A good demonstration is that attack vector *iii* will cause the framework to provide the output FAIL while vector *v* always results in the output PASS for the inputs $\phi = \phi_{subset}$ and $\mathcal{T} = \{0..63\}$. This result is expected because *iii* and *v* use the same attacker definition, but their trust relationships differ. In vector *iii* all private keys have been compromised, vector *v* has no compromised keys. The compromised keys allow vector *iii* to modify control messages and provide the corresponding signature. The results of these two validation runs are provided in Chapter 4 and correspond to the expected outcomes.

In addition to validating the two new functions it is also necessary to validate that Secure OLSR does not affect the original validation. A validation run with $A = i$, $\phi = \phi_{complete}$, and $\mathbf{T} = \{0..63\}$ should provide sufficient evidence that the Secure OLSR model behaves no differently than the original. In this case the output should be PASS for all inputs. Chapter 4 captures the results which validate Secure OLSR.

3.6.3 Security Verification. Security verification can be performed exhaustively over all inputs, $\mathbf{T} = \{0..63\}$, $\phi = \phi_{subset}$, $\mathbf{A} = \{i..v\}$. Any violations of ϕ will reveal the counter-example showing the steps leading to the property violation. The state-space required for the set of topologies $\mathbf{T} = \{64..1023\}$ (i.e., all five-node networks) is too great for performing an exhaustive verification. As a result, only estimates of the full state-space search are possible. This non-exhaustive verification is performed by setting the verifier's BITSTATE compilation flag. Using the bit-state approximation should reveal most security violations and provide the appropriate counter-example; however, this does not guarantee that a violation will be found even if it exists for the specified inputs. The Secure OLSR model understands the trust relationships, $\mathcal{T}$, provided in the attack vectors. For this reason attack vectors with no compromised keys should be unable to add false routes. Any findings to the contrary will reveal an unknown attack against Secure OLSR.

The expected outcomes for each attack vector are presented below. Results for all scenarios are available in Chapter 4.

Attack *i* No attackers are defined, the framework should always output PASS.

Attack *ii* The attacker contains a single compromised identity. With this identity the attacker generates arbitrary TC messages that contain false routes. Since the identity of the attacker is trusted nodes in the network will accept the false routes introduced by the attacker.

Attack *iii* The attacker contains compromised identities for all node identifiers. Given this information the attacker modifies and signs TC messages that it is either elected to forward or generates to advertise itself as an MPR. Because of the compromised identities and attacker definition the framework should output FAIL whenever the attacker node would be selected as an MPR.

Attack *iv* Although no identities are compromised Secure OLSR is unable to defend against relay attacks. Such attacks are independent of the current trust relationship because the malicious node does not attempt to modify any relayed control traffic. An output of FAIL is expected any time the attacker joins two benign nodes that share no common links.

Attack *v* The attacker definition is equivalent to *iii* but no identities are compromised. The security features of Secure OLSR are expected to prevent the injection of any false routes so the framework should always output PASS for this attack vector.

3.7 Chapter Summary

This chapter on the research methodology has presented the goals and assumptions of the security verification framework for secure MANET routing protocols. A justification of the framework's core model checker, SPIN was given. The design of the framework was presented along with techniques for improving the tractability of MANET routing protocol models. Finally, both OLSR and Secure OLSR models were built and the validation and verification strategies were motivated for each model.

4 Results

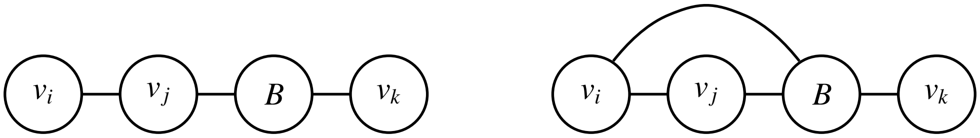
A validation and verification strategy was proposed in Section 3.5 for the OLSR protocol. This chapter presents the results of this validation and verification and compares them with the expected outcomes. The chapter also provides the results for the validation and verification of Secure OLSR as discussed in Section 3.6. These results are compared to the expected outcomes for each experiment.

4.1 OLSR Validation and Verification

Exhaustive verification of the OLSR model for inputs $\mathbf{I} = \{A = i$, $\mathbf{T} = \{0..63\}$, and $\phi = \phi_{complete}\}$ returns the string *PASS* (i.e., no security violation) for all $T \in \mathbf{T}$. All benign network topologies in $\mathbf{T}$ are considered secure when no attacker is present. The result proves that for the verified topologies the model never violates $\phi_{complete}$; thus, the model provides the same neighbor sensing and topology control that would be expected from any node in an OLSR implementation. These results demonstrate that the OLSR model is valid (see Section 3.3.1).

For the inputs $\mathbf{I} = \{A = i, \mathbf{T} = \{0..63\}, \phi = \phi_{subset}\}$, an exhaustive verification is performed. The framework provides the output of PASS for all $\mathbf{I}$, which is the expected result since no malicious nodes are introduced. Next $\mathbf{I} = \{A = v, \mathbf{T} = \{0..63\}, \phi = \phi_{subset}\}$ is tested. Here, the attack vector, $A = v$, represents a single attacker that replaces the last node, v_{N-1}, with a malicious node. This malicious node adds false routes to all TC messages it forwards or generates. Exhaustive verification over all inputs in $\mathbf{I}$ returns *FAIL* (i.e., ϕ_{subset} is violated) under topologies 26, 28, 30, 41, 44, 45, 49, 50, and 51. The configuration of the failing topologies is generalized by Figure 4.1a. The malicious node, B, advertises false routes to its neighbors when it has been elected as an MPR by v_j or v_k. In addition, v_j advertises that it is an MPR for v_i but B modifies this advertisement before forwarding. The property ϕ_{subset} detects this because the Topology Control set of v_k is eventually inconsistent with the Topology Control set of v_j.

Continuing with input $\mathbf{I} = \{A = v, \mathbf{T} = \{0..63\}, \phi = \phi_{subset}\}$, the verification reports that topologies 57, 58, 60, ... return *PASS* (i.e., the protocol is secure for the inputs). Figure 4.1b represents the general case for these topologies. Node *B* is elected as an MPR by all benign nodes, and *B* always modifies this information by advertising that it has a link to all benign nodes. It follows that v_i, v_j, and v_k have consistent TC sets; therefore, no property violation is found because *B*'s advertised routes exist. The failure to detect a malicious route in this case is acceptable because the goal of analyzing security is only to show if *B* can add an inaccurate route to the network.



(a) T=26, 28, 30, 41, 44, 45, 49, 50, 51

(b) T=57, 58, 60, ...

Figure 4.1: Generic networks where $i \neq j \neq k$ and *B* is malicious.

4.2 Secure OLSR Validation

The attack vectors can be applied in order to show that the model of the Secure OLSR protocol accurately reflects the protocol's specifications. Specifically, attack vectors *i*, *iii*, and *v* are used for model validation against the Secure OLSR specification, as discussed in Section 3.6.

The attack vectors *i* and *v* are expected not to affect the specification's operation, and the model should reflect this behavior. Clearly, the cases where attack vector *i* is applied will have no impact on the model because no malicious nodes are defined. When vector *v* is applied the model will block all untrusted nodes. Against *i* and *v*, the protocol should behave no differently than it does in OLSR.

Attack vector *iii* reflects what happens when the fundamental cryptographic assumption of private keys fail, the expected result is that an attacker can compromise any node's routing information. When vector *iii* is applied, the model will be vulnerable to any actions of the attacker because it can forge messages for any benign identity. In this case, the attacker only adds false routes to TC messages. Against *iii*, the protocol is compromised and should be completely vulnerable.

With these three attack vectors, it is possible demonstrate that the Secure OLSR model performs in accordance with its specifications. Attack vectors *i*, *iii*, and *v* validate the fundamental mechanism used for securing OLSR. Vectors *i* and *v* should be thwarted by the security mechanism; vector *iii* should fail in some instances because it breaks the cryptographic premise that private keys are private.

Under attack vector *iii*, an attacker with knowledge of all private keys is shown to add inaccurate routes to a Secure OLSR network. With inputs $A = iii$, $\mathbf{T} = \{0..63\}$, and $\phi = \phi_{subset}$ the verification demonstrates that in a subset of **T** the attacker successfully provides inaccurate routes, as shown in Table 4.1. Inaccurate routes are only found in topologies where the attacker is elected as an MPR. The property, ϕ_{subset}, reveals that Secure OLSR is vulnerable to attack vector *iii*. This result is expected because the vector violates the premise that private keys are kept private.

In attack vector *v*, the attacker modifies TC messages before transmission, but fails to produce valid signatures for the modifications. Since the signatures are invalid all modified messages should be ignored. The inputs $A = v$, $\mathbf{T} = \{0..63\}$, and $\phi = \phi_{subset}$ are tested to show that false routes cannot be introduced. The results, in Table 4.2, show that no inaccurate routes are added to any topologies in **T** by attack *v*. From these results it is

Table 4.1: Verification against attack *iii*: Omnipotent

T	ϕ_{subset}	T	ϕ_{subset}	T	ϕ_{subset}	T	ϕ_{subset}
0	PASS	16	PASS	32	PASS	48	PASS
1	PASS	17	PASS	33	PASS	**49**	FAIL
2	PASS	18	PASS	34	PASS	**50**	FAIL
3	PASS	19	PASS	35	PASS	**51**	FAIL
4	PASS	20	PASS	36	PASS	52	PASS
5	PASS	21	PASS	37	PASS	53	PASS
6	PASS	22	PASS	38	PASS	54	PASS
7	PASS	23	PASS	39	PASS	55	PASS
8	PASS	24	PASS	40	PASS	56	PASS
9	PASS	25	PASS	**41**	FAIL	57	PASS
10	PASS	**26**	FAIL	42	PASS	58	PASS
11	PASS	27	PASS	43	PASS	59	PASS
12	PASS	**28**	FAIL	**44**	FAIL	60	PASS
13	PASS	29	PASS	**45**	FAIL	61	PASS
14	PASS	**30**	FAIL	46	PASS	62	PASS
15	PASS	31	PASS	47	PASS	63	PASS

concluded Secure OLSR prevents outside attackers from adding false routes for the specified set of inputs.

Using inputs $A = i$, $\mathbf{T} = \{0..63\}$, and $\phi = \phi_{complete}$ the verification reveals no violations. Since every node generates control messages that are signed correctly, then every node should only receive messages that are *valid*. Applying $\phi = \phi_{subset}$ yields no violations, as shown in Table 4.3. This behavior is equivalent to that of the OLSR model; therefore, the results show that by implementing Secure OLSR the underlying behavior of OLSR is not changed. Table 4.3 provides a summary of the results given $A = i$, $\mathbf{T} = \{0..63\}$, and $\phi = \phi_{complete}$ and $A = i$, $\mathbf{T} = \{0..63\}$, and $\phi = \phi_{subset}$.

4.3 Secure OLSR Verification

This section presents the verification of Secure OLSR's security properties. It is necessary to define the model's inputs (A, T, and ϕ). A is any attacker vector *i* through *v*

Table 4.2: Verification against attack vector *v*: Outsider

T	ϕ_{subset}	T	ϕ_{subset}	T	ϕ_{subset}	T	ϕ_{subset}
0	PASS	16	PASS	32	PASS	48	PASS
1	PASS	17	PASS	33	PASS	49	PASS
2	PASS	18	PASS	34	PASS	50	PASS
3	PASS	19	PASS	35	PASS	51	PASS
4	PASS	20	PASS	36	PASS	52	PASS
5	PASS	21	PASS	37	PASS	53	PASS
6	PASS	22	PASS	38	PASS	54	PASS
7	PASS	23	PASS	39	PASS	55	PASS
8	PASS	24	PASS	40	PASS	56	PASS
9	PASS	25	PASS	41	PASS	57	PASS
10	PASS	26	PASS	42	PASS	58	PASS
11	PASS	27	PASS	43	PASS	59	PASS
12	PASS	28	PASS	44	PASS	60	PASS
13	PASS	29	PASS	45	PASS	61	PASS
14	PASS	30	PASS	46	PASS	62	PASS
15	PASS	31	PASS	47	PASS	63	PASS

Table 4.3: Verification against attack *i*: Benign

T	$\phi_{complete}, \phi_{subset}$	T	$\phi_{complete}, \phi_{subset}$	T	$\phi_{complete}, \phi_{subset}$	T	$\phi_{complete}, \phi_{subset}$
0	PASS	16	PASS	32	PASS	48	PASS
1	PASS	17	PASS	33	PASS	49	PASS
2	PASS	18	PASS	34	PASS	50	PASS
3	PASS	19	PASS	35	PASS	51	PASS
4	PASS	20	PASS	36	PASS	52	PASS
5	PASS	21	PASS	37	PASS	53	PASS
6	PASS	22	PASS	38	PASS	54	PASS
7	PASS	23	PASS	39	PASS	55	PASS
8	PASS	24	PASS	40	PASS	56	PASS
9	PASS	25	PASS	41	PASS	57	PASS
10	PASS	26	PASS	42	PASS	58	PASS
11	PASS	27	PASS	43	PASS	59	PASS
12	PASS	28	PASS	44	PASS	60	PASS
13	PASS	29	PASS	45	PASS	61	PASS
14	PASS	30	PASS	46	PASS	62	PASS
15	PASS	31	PASS	47	PASS	63	PASS

defined previously. T is a topology between 0 and 63, and ϕ is the security property captured by ϕ_{subset}. For each combination of these inputs the model verification produces the output *PASS* or *FAIL*. The output PASS refers to the verification that finds no security violations, and FAIL refers to one in which security violations are found.

The results of Attack *i*, where all nodes participating in the network are benign, is shown in Table 4.3. The results, as expected, show that ϕ_{subset} holds for all possible topologies. Secure OLSR meets the same routing objectives as OLSR when there are no adversaries.

When attack vector *ii* is tested in the framework it reveals a significant number FAIL outputs. Under attack *ii* the attacker definition continuously advertises TC messages with false routes. The only false routes it advertises are from the attack node v_{N-1} to nodes v_0, v_1, and v_2. There are some cases when $v_{N-1} = v_2$. The output of PASS, in Table 4.4, is explained by two scenarios. The first is when the attacker definition actually advertises routes that exist, this occurs in topologies 1, 6, and 56-63. The second scenario is when node v_0 is not connected to the larger network, the attacker is ineffective because it cannot send any control traffic to v_0. In this last scenario v_0 accurately believes it is an isolated node. This scenario accounts for all remaining PASS outputs in Table 4.4.

Attack *iii* shows the case where the attacker chooses only to modify the TC messages which it has been elected to forward in the network. Some topologies verified against this attack are vulnerable while others are not, as presented in Table 4.1. Since the private keys are compromised the attacker is able to modify TC messages that are accepted by all benign nodes. The same analysis covered for Figure 4.1 also applies in the case of Secure OLSR.

Applying attack vector *iv* reveals that 19 topologies lead to an output of FAIL, see Table 4.5. The attack expressed here is the invisible node attack. The failures correspond to cases where the invisible node is able to add non-existent links to the network. This

Table 4.4: Verification against attack *ii*: Byzantine

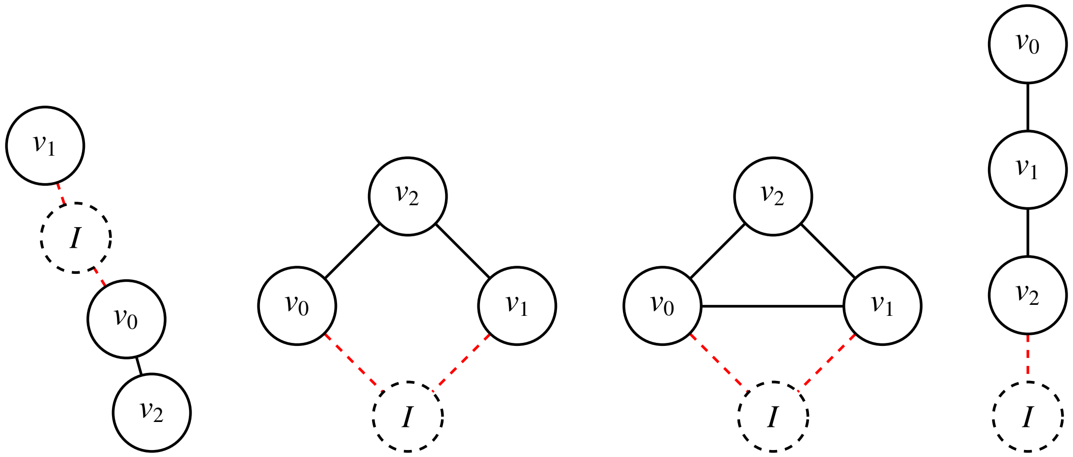
T	ϕ_{subset}	T	ϕ_{subset}	T	ϕ_{subset}	T	ϕ_{subset}
0	PASS	16	PASS	32	PASS	48	PASS
1	PASS	**17**	FAIL	33	PASS	**49**	FAIL
2	FAIL	18	PASS	**34**	FAIL	**50**	FAIL
3	FAIL	**19**	FAIL	**35**	FAIL	**51**	FAIL
4	PASS	20	PASS	36	PASS	52	PASS
5	FAIL	**21**	FAIL	**37**	FAIL	**53**	FAIL
6	PASS	**22**	FAIL	**38**	FAIL	**54**	FAIL
7	FAIL	**23**	FAIL	**39**	FAIL	**55**	FAIL
8	FAIL	**24**	FAIL	**40**	FAIL	56	PASS
9	FAIL	**25**	FAIL	**41**	FAIL	57	PASS
10	FAIL	**26**	FAIL	**42**	FAIL	58	PASS
11	FAIL	**27**	FAIL	**43**	FAIL	59	PASS
12	FAIL	**28**	FAIL	**44**	FAIL	60	PASS
13	FAIL	**29**	FAIL	**45**	FAIL	61	PASS
14	FAIL	**30**	FAIL	**46**	FAIL	62	PASS
15	FAIL	**31**	FAIL	**47**	FAIL	63	PASS

attack is able to violate the definition of security by introducing false routes. Although, not all cases lead to a security violation, the attack is effective under certain topological configurations. Figure 4.2 demonstrates how some topological configurations will lead to the output of FAIL. In general, the output FAIL occurs if the invisible node adds a link between two nodes that are not otherwise connected.

When faced with attack *v*, Secure OLSR ignores all messages signed by the attacking node because its identity is in the set of untrusted nodes ($v_{N-1} \in \mathbb{U}$). The verification run against this attack vector demonstrates the expected result that *v* is an ineffective attack. Table 4.2 shows that the protocol is not vulnerable to attack vector *v* for the set of topologies examined. This result was expected under this attack and demonstrates that Secure OLSR adds security for the set of inputs examined.

Table 4.5: Verification against attack *iv*: Invisible node

T	ϕ_{subset}	T	ϕ_{subset}	T	ϕ_{subset}	T	ϕ_{subset}
0	PASS	16	PASS	32	PASS	48	PASS
1	PASS	17	PASS	33	PASS	**49**	FAIL
2	PASS	18	PASS	34	PASS	**50**	FAIL
3	PASS	19	PASS	35	PASS	**51**	FAIL
4	PASS	20	PASS	36	PASS	52	PASS
5	PASS	21	PASS	37	PASS	53	PASS
6	FAIL	22	PASS	38	PASS	54	PASS
7	PASS	23	PASS	39	PASS	55	PASS
8	PASS	**24**	FAIL	**40**	FAIL	**56**	FAIL
9	PASS	25	PASS	**41**	FAIL	**57**	FAIL
10	PASS	**26**	FAIL	42	PASS	**58**	FAIL
11	PASS	27	PASS	43	PASS	**59**	FAIL
12	PASS	**28**	FAIL	**44**	FAIL	**60**	FAIL
13	PASS	29	PASS	**45**	FAIL	**61**	FAIL
14	PASS	**30**	FAIL	46	PASS	**62**	FAIL
15	PASS	31	PASS	47	PASS	63	PASS



(a) FAIL

(b) FAIL

(c) PASS

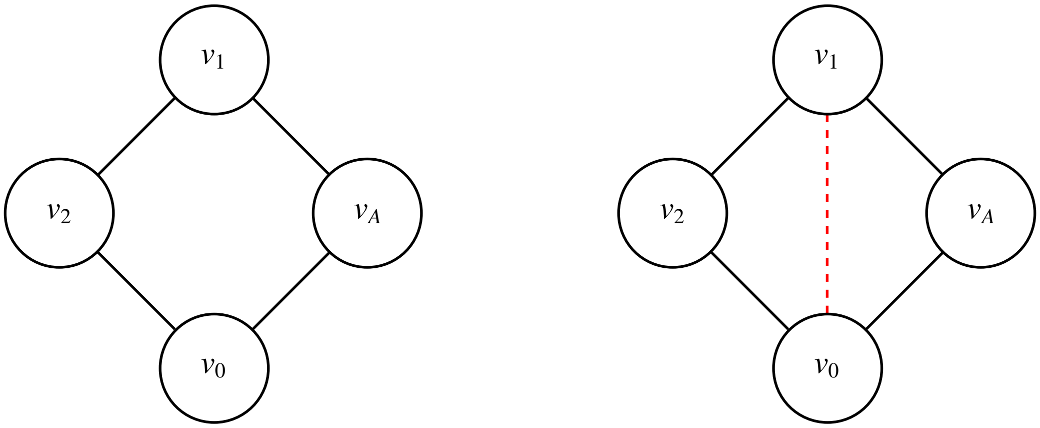
(d) PASS

Figure 4.2: Collection of topologies involving an invisible node, I. *FAIL* implies I adds at least one false route. *PASS* implies I has no effect.

4.4 The Subtlety of Attacks

In the analysis of the invisible node attack, vector *iv*, it would be insufficient to randomly pick a few topologies to verify. For example, had topologies 31 and 37 (Figure 4.2c and 4.2d) been selected the results would show that the invisible node had no impact. Instead, it is necessary to examine every topology to show which are vulnerable. For attack vector *iv* an obvious rule can be stated: *if* v_I *is a neighbor of two or more nodes which do not share a link, then the attacker adds a false route.* The rule for the invisible node attack is easily derived even without the verification of the Secure OLSR model. The important result is that attacks do not apply equally in all network topologies. New attacks may not lend themselves to such simple rules; therefore, it is essential for the verification framework to examine *all possible* node configurations.

The attack vector *ii* causes property violations in all topologies verified. The vector demonstrates that if just one private key is compromised, or there is no security, then any configuration can be forced to accept inaccurate routes. A less obtrusive attack vector (e.g., vector *iii*) limits the attacker, allowing it to only modify TC messages which it is responsible for forwarding. The results of such an attack are shown in Table 4.1. In configurations where the malicious node, v_A, is selected as an MPR by at least one node the property, ϕ_{subset}, is violated. Recall that MPR selection is non-deterministic. It follows that the attack will not work in network scenarios where v_A is not elected as an MPR. An unsuccessful example of this attack is illustrated in Figure 4.3a. Figure 4.3b provides an example of a successful attack. These examples are different because 4.3a did not select v_A as an MPR whereas 4.3b did. The attacker must be selected as the MPR to be successful, so the attack succeeded in this instance. Two SPIN simulations provided these different results under the same inputs. See Appendix A.2 and A.3 for the full message sequence charts associated with Figures 4.3a and 4.3b. This analysis underscores the need to perform exhaustive verification when searching for security flaws in a distributed system. The automatic discovery of attacks against secure, table-based protocols is dependent on the ability to exhaustively search a protocol's possible interactions in order to disclose subtle sequences that lead to property violations.



(a) Network view is accurately built, despite attack vector *iii*.

(b) The link v_1-v_0 is an inaccuracy added by attack vector *iii*.

Figure 4.3: Topology #30, under attack vector *iii*, in two separate simulation results.

4.5 Chapter Summary

The results of the validation and verification experiments designed in Chapter 3 were tested. These results demonstrate that the OLSR and Secure OLSR models correctly reflect their specifications. No new attacks were revealed by applying the automatic verification framework; however, the verification under the set of existing attacks proves that the framework has the ability to detect subtle errors in proactive, table-based routing protocols.

5 Conclusions

The prevalence of wireless communications makes it a low hanging fruit for innovative products as well as unscrupulous hackers with malicious intent. This research advances the state of an automated verification framework into the realm of proactive ad hoc routing protocols. Although no method may ever be able to prevent the subversion of routing protocols, it is very important to understand the security vulnerabilities that exist.

5.1 Research Problem

In today's network applications, end-to-end security is often achieved through public key encryption. At this level, however, the only guarantees that can be made are that a user's data is unmodified, remains private, or both. Application level security cannot guarantee the delivery of data. Thus, at the routing protocol layer a different level of security must be introduced to protect availability. This need is especially true in wireless networks like MANETs.

Attacks against routing in MANETs are often subtle because of the wide range of actions that an attacker may employ. For example, a relaying attack allows an adversary to take control of data paths by simply causing a shorter path to be advertised through it. An important step to preventing attacks against a wireless routing protocol is to understand the possible attack avenues and the particular attack sequences that are available to an attacker.

5.2 Contributions

This research provides a modeling framework capable of exhaustively verifying security properties in MANET table-based routing protocols. The research directly extends a similar framework developed in Andel [2].

Specifically, this framework adds support for table-based protocols, and in doing so advanced several formal methods modeling concepts for MANETs. Linear temporal logic was added, and scalability was improved with the use of the macro processor *m4*. The concept of the wireless medium was improved with greater scalability, tractability, and abstraction. Finally, the universal, table-based verification property, ϕ_{subset}, was developed and it is suggested that this property can be used in the verification of all table-based protocols.

LTL is introduced to the framework and enables a property to be evaluated in all steps of execution, this is the first application of LTL to secure, table-based routing protocols. The macro processor *m4* is applied to the framework and provides size scalability, greater expressiveness, and adaptable attack vectors. The use of *m4* removes verification dependence on external tools or scripting languages and centralizes the modeling code.

The wireless medium concept is further developed to improve the abstraction, tractability, and size scalability. Specifically the wireless medium server is designed such that it can be used for modeling any ad hoc routing protocol. This is valuable because new protocols will not rely on the implementation of the wireless medium model.

Verification of Secure OLSR against several attack vectors confirms the findings of previously discovered attacks. Although the verification framework did not reveal any undiscovered attacks against Secure OLSR, this does not mean that no other attacks against the protocol exist. It only means that the attack vectors, topologies, and properties tested did not reveal any unknown attack sequences.

5.3 Future Work

The framework presented here is an extension into the realm of proactive ad hoc routing protocols. This research has exposed many more avenues of research needed for the automatic verification of secure routing protocols.

Several of the advances developed are directly applicable to the prior research in accumulation-based, reactive routing protocols. For example, the use of *m4* can be used to improve scalability and improve the readability of the Promela models. The wireless medium changes can also be applied to greatly reduce the state-space required in verification runs. Applying the concepts learned in this research would allow the framework developed in Andel [2], to be applied to networks containing many more nodes.

A clear path forward with this framework is to apply it to other secure, table-based protocols such as SLSP and SAODV. In the case of on-demand protocols the property, ϕ_{subset}, is still applicable. It may even be desirable to force all nodes in table-based routing protocols to send route requests. The final decision is, however, in the hands of the designer.

Proactive, table-based protocols can now be modeled for exhaustive verification. The next logical step is to add support for secure hybrid protocols. Hybrid protocols utilize both reactive and proactive routing strategies in a single protocol.

Although great improvements were made in the wireless medium, its current implementation may be improved. One improvement is to remove the limitation that imposes finite execution sequences.

The verifications never found any undiscovered attack sequences in Secure OLSR. This absence is not to say that an undiscovered attack does or does not exist. More sophisticated attack vectors need to be defined. These attack vectors must assume that no cryptographic material is compromised and will be more advanced than relay attacks. A successful attack vector may incorporate relaying and other mechanisms to attempt to trick the protocol into accepting false routes. If, for example, an adversary is able to trick another node into signing a message on its behalf then several possible attacks may become possible. These attack vectors can then be used to automatically discover the associated attack sequences and their applicable topologies.

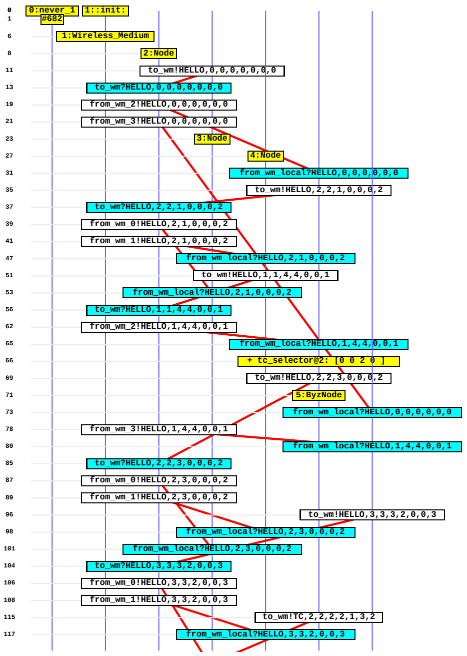
Appendix A: Message Sequence Charts

A.1 Description

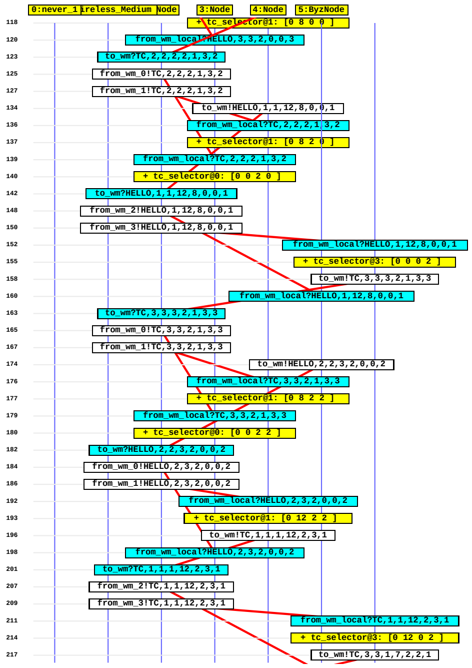
The following message sequence charts describe the series of messages exchanged between nodes. Section A.2 shows how the configuration of topology 30 leads to an inaccurate routing decision as discussed in Chapter 4. Section A.3 demonstrates a sequence of events where attack vector *iii* fails to affect the appropriate routing decisions for topology 30.

A.2 Topology 30: Inaccurate Route

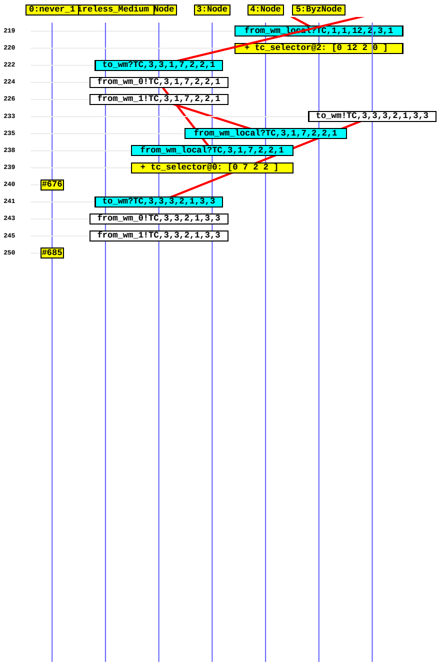
Spin Version 6.1.0 -- 4 May 2011 -- solsr.pml -- MSC -- 1


0 0:never_1 1::init:
1 #682
6 1:Wireless_Medium
8 2:Node
11 to_wm!HELLO,0,0,0,0,0,0,0
13 to_wm?HELLO,0,0,0,0,0,0,0
19 from_wm_2!HELLO,0,0,0,0,0,0
21 from_wm_3!HELLO,0,0,0,0,0,0
23 3:Node
27 4:Node
31 from_wm_local?HELLO,0,0,0,0,0,0
35 to_wm!HELLO,2,2,1,0,0,0,2
37 to_wm?HELLO,2,2,1,0,0,0,2
39 from_wm_0!HELLO,2,1,0,0,0,2
41 from_wm_1!HELLO,2,1,0,0,0,2
47 from_wm_local?HELLO,2,1,0,0,0,2
51 to_wm!HELLO,1,1,4,4,0,0,1
53 from_wm_local?HELLO,2,1,0,0,0,2
56 to_wm?HELLO,1,1,4,4,0,0,1
62 from_wm_2!HELLO,1,4,4,0,0,1
65 from_wm_local?HELLO,1,4,4,0,0,1
66 + tc_selector@2: [0 0 2 0]
69 to_wm!HELLO,2,2,3,0,0,0,2
71 5:ByzNode
73 from_wm_local?HELLO,0,0,0,0,0,0
78 from_wm_3!HELLO,1,4,4,0,0,1
80 from_wm_local?HELLO,1,4,4,0,0,1
85 to_wm?HELLO,2,2,3,0,0,0,2
87 from_wm_0!HELLO,2,3,0,0,0,2
89 from_wm_1!HELLO,2,3,0,0,0,2
96 to_wm!HELLO,3,3,3,2,0,0,3
98 from_wm_local?HELLO,2,3,0,0,0,2
101 from_wm_local?HELLO,2,3,0,0,0,2
104 to_wm?HELLO,3,3,3,2,0,0,3
106 from_wm_0!HELLO,3,3,2,0,0,3
108 from_wm_1!HELLO,3,3,2,0,0,3
115 to_wm!TC,2,2,2,2,1,3,2
117 from_wm_local?HELLO,3,3,2,0,0,3

Spin Version 6.1.0 -- 4 May 2011 -- solsr.pml -- MSC -- 2

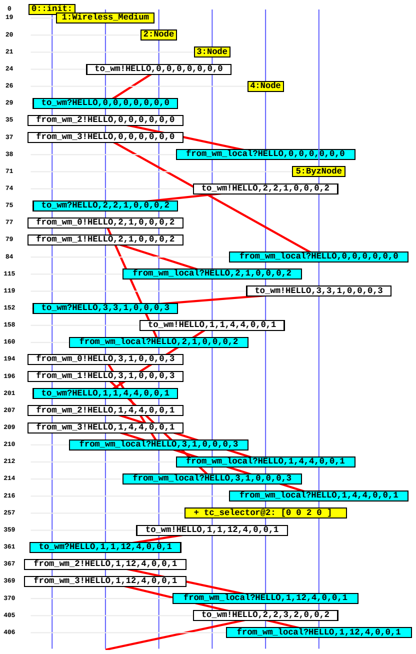

0:never_1 ireless_Medium Node 3:Node 4:Node 5:ByzNode
118 + tc_selector@1: [0 8 0 0]
120 from_wm_local?HELLO,3,3,2,0,0,3
123 to_wm?TC,2,2,2,2,1,3,2
125 from_wm_0!TC,2,2,2,1,3,2
127 from_wm_1!TC,2,2,2,1,3,2
134 to_wm!HELLO,1,1,12,8,0,0,1
136 from_wm_local?TC,2,2,2,1,3,2
137 + tc_selector@1: [0 8 2 0]
139 from_wm_local?TC,2,2,2,1,3,2
140 + tc_selector@0: [0 0 2 0]
142 to_wm?HELLO,1,1,12,8,0,0,1
148 from_wm_2!HELLO,1,12,8,0,0,1
150 from_wm_3!HELLO,1,12,8,0,0,1
152 from_wm_local?HELLO,1,12,8,0,0,1
155 + tc_selector@3: [0 0 0 2]
158 to_wm!TC,3,3,3,2,1,3,3
160 from_wm_local?HELLO,1,12,8,0,0,1
163 to_wm?TC,3,3,3,2,1,3,3
165 from_wm_0!TC,3,3,2,1,3,3
167 from_wm_1!TC,3,3,2,1,3,3
174 to_wm!HELLO,2,2,3,2,0,0,2
176 from_wm_local?TC,3,3,2,1,3,3
177 + tc_selector@1: [0 8 2 2]
179 from_wm_local?TC,3,3,2,1,3,3
180 + tc_selector@0: [0 0 2 2]
182 to_wm?HELLO,2,2,3,2,0,0,2
184 from_wm_0!HELLO,2,3,2,0,0,2
186 from_wm_1!HELLO,2,3,2,0,0,2
192 from_wm_local?HELLO,2,3,2,0,0,2
193 + tc_selector@1: [0 12 2 2]
196 to_wm!TC,1,1,1,12,2,3,1
198 from_wm_local?HELLO,2,3,2,0,0,2
201 to_wm?TC,1,1,1,12,2,3,1
207 from_wm_2!TC,1,1,12,2,3,1
209 from_wm_3!TC,1,1,12,2,3,1
211 from_wm_local?TC,1,1,12,2,3,1
214 + tc_selector@3: [0 12 0 2]
217 to_wm!TC,3,3,1,7,2,2,1

Spin Version 6.1.0 -- 4 May 2011 -- solsr.pml -- MSC -- 3

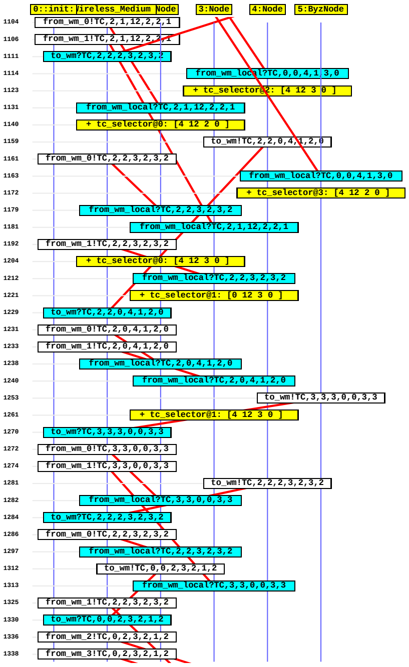

0:never_1 ireless_Medium Node 3:Node 4:Node 5:ByzNode
219 from_wm_local?TC,1,1,12,2,3,1
220 + tc_selector@2: [0 12 2 0]
222 to_wm?TC,3,3,1,7,2,2,1
224 from_wm_0!TC,3,1,7,2,2,1
226 from_wm_1!TC,3,1,7,2,2,1
233 to_wm!TC,3,3,3,2,1,3,3
235 from_wm_local?TC,3,1,7,2,2,1
238 from_wm_local?TC,3,1,7,2,2,1
239 + tc_selector@0: [0 7 2 2]
240 #676
241 to_wm?TC,3,3,3,2,1,3,3
243 from_wm_0!TC,3,3,2,1,3,3
245 from_wm_1!TC,3,3,2,1,3,3
250 #685

A.3 Topology 30: Accurate Route

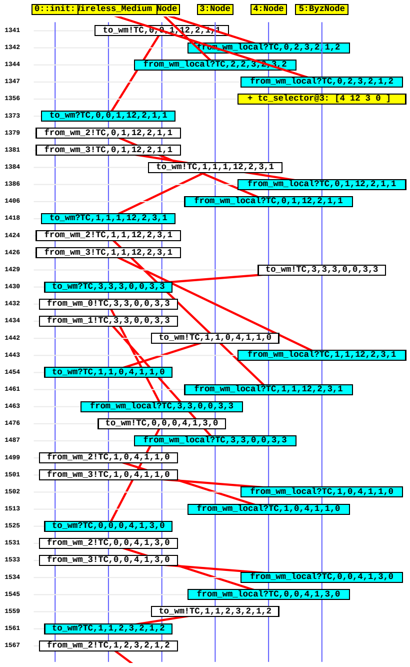
Spin Version 6.1.0 -- 4 May 2011 -- solsr.pml -- MSC -- 1


0 0::init:
19 1:Wireless_Medium
20 2:Node
21 3:Node
24 to_wm!HELLO,0,0,0,0,0,0,0
26 4:Node
29 to_wm?HELLO,0,0,0,0,0,0,0
35 from_wm_2!HELLO,0,0,0,0,0,0
37 from_wm_3!HELLO,0,0,0,0,0,0
38 from_wm_local?HELLO,0,0,0,0,0,0
71 5:ByzNode
74 to_wm!HELLO,2,2,1,0,0,0,2
75 to_wm?HELLO,2,2,1,0,0,0,2
77 from_wm_0!HELLO,2,1,0,0,0,2
79 from_wm_1!HELLO,2,1,0,0,0,2
84 from_wm_local?HELLO,0,0,0,0,0,0
115 from_wm_local?HELLO,2,1,0,0,0,2
119 to_wm!HELLO,3,3,1,0,0,0,3
152 to_wm?HELLO,3,3,1,0,0,0,3
158 to_wm!HELLO,1,1,4,4,0,0,1
160 from_wm_local?HELLO,2,1,0,0,0,2
194 from_wm_0!HELLO,3,1,0,0,0,3
196 from_wm_1!HELLO,3,1,0,0,0,3
201 to_wm?HELLO,1,1,4,4,0,0,1
207 from_wm_2!HELLO,1,4,4,0,0,1
209 from_wm_3!HELLO,1,4,4,0,0,1
210 from_wm_local?HELLO,3,1,0,0,0,3
212 from_wm_local?HELLO,1,4,4,0,0,1
214 from_wm_local?HELLO,3,1,0,0,0,3
216 from_wm_local?HELLO,1,4,4,0,0,1
257 + tc_selector@2: [0 0 2 0]
359 to_wm!HELLO,1,1,12,4,0,0,1
361 to_wm?HELLO,1,1,12,4,0,0,1
367 from_wm_2!HELLO,1,12,4,0,0,1
369 from_wm_3!HELLO,1,12,4,0,0,1
370 from_wm_local?HELLO,1,12,4,0,0,1
405 to_wm!HELLO,2,2,3,2,0,0,2
406 from_wm_local?HELLO,1,12,4,0,0,1

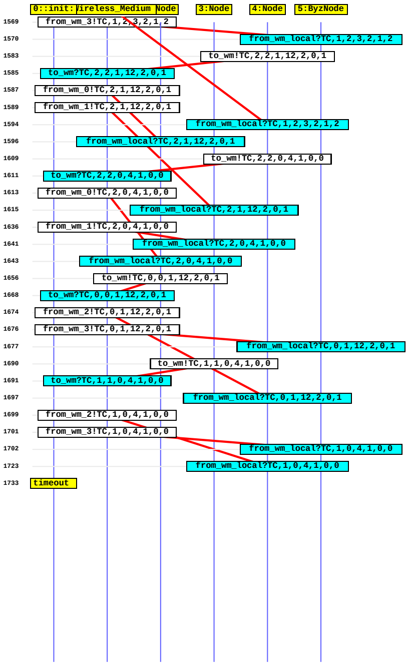
Spin Version 6.1.0 -- 4 May 2011 -- solsr.pml -- MSC -- 4


0::init:
Wireless_Medium
Node
3:Node
4:Node
5:ByzNode
1104 from_wm_0!TC,2,1,12,2,2,1
1106 from_wm_1!TC,2,1,12,2,2,1
1111 to_wm?TC,2,2,2,3,2,3,2
1114 from_wm_local?TC,0,0,4,1,3,0
1123 + tc_selector@2: [4 12 3 0]
1131 from_wm_local?TC,2,1,12,2,2,1
1140 + tc_selector@0: [4 12 2 0]
1159 to_wm!TC,2,2,0,4,1,2,0
1161 from_wm_0!TC,2,2,3,2,3,2
1163 from_wm_local?TC,0,0,4,1,3,0
1172 + tc_selector@3: [4 12 2 0]
1179 from_wm_local?TC,2,2,3,2,3,2
1181 from_wm_local?TC,2,1,12,2,2,1
1192 from_wm_1!TC,2,2,3,2,3,2
1204 + tc_selector@0: [4 12 3 0]
1212 from_wm_local?TC,2,2,3,2,3,2
1221 + tc_selector@1: [0 12 3 0]
1229 to_wm?TC,2,2,0,4,1,2,0
1231 from_wm_0!TC,2,0,4,1,2,0
1233 from_wm_1!TC,2,0,4,1,2,0
1238 from_wm_local?TC,2,0,4,1,2,0
1240 from_wm_local?TC,2,0,4,1,2,0
1253 to_wm!TC,3,3,3,0,0,3,3
1261 + tc_selector@1: [4 12 3 0]
1270 to_wm?TC,3,3,3,0,0,3,3
1272 from_wm_0!TC,3,3,0,0,3,3
1274 from_wm_1!TC,3,3,0,0,3,3
1281 to_wm!TC,2,2,2,3,2,3,2
1282 from_wm_local?TC,3,3,0,0,3,3
1284 to_wm?TC,2,2,2,3,2,3,2
1286 from_wm_0!TC,2,2,3,2,3,2
1297 from_wm_local?TC,2,2,3,2,3,2
1312 to_wm!TC,0,0,2,3,2,1,2
1313 from_wm_local?TC,3,3,0,0,3,3
1325 from_wm_1!TC,2,2,3,2,3,2
1330 to_wm?TC,0,0,2,3,2,1,2
1336 from_wm_2!TC,0,2,3,2,1,2
1338 from_wm_3!TC,0,2,3,2,1,2

Spin Version 6.1.0 -- 4 May 2011 -- solsr.pml -- MSC -- 5


0::init:
Wireless_Medium
Node
3:Node
4:Node
5:ByzNode
1341 to_wm!TC,0,0,1,12,2,1,1
1342 from_wm_local?TC,0,2,3,2,1,2
1344 from_wm_local?TC,2,2,3,2,3,2
1347 from_wm_local?TC,0,2,3,2,1,2
1356 + tc_selector@3: [4 12 3 0]
1373 to_wm?TC,0,0,1,12,2,1,1
1379 from_wm_2!TC,0,1,12,2,1,1
1381 from_wm_3!TC,0,1,12,2,1,1
1384 to_wm!TC,1,1,1,12,2,3,1
1386 from_wm_local?TC,0,1,12,2,1,1
1406 from_wm_local?TC,0,1,12,2,1,1
1418 to_wm?TC,1,1,1,12,2,3,1
1424 from_wm_2!TC,1,1,12,2,3,1
1426 from_wm_3!TC,1,1,12,2,3,1
1429 to_wm!TC,3,3,3,0,0,3,3
1430 to_wm?TC,3,3,3,0,0,3,3
1432 from_wm_0!TC,3,3,0,0,3,3
1434 from_wm_1!TC,3,3,0,0,3,3
1442 to_wm!TC,1,1,0,4,1,1,0
1443 from_wm_local?TC,1,1,12,2,3,1
1454 to_wm?TC,1,1,0,4,1,1,0
1461 from_wm_local?TC,1,1,12,2,3,1
1463 from_wm_local?TC,3,3,0,0,3,3
1476 to_wm!TC,0,0,0,4,1,3,0
1487 from_wm_local?TC,3,3,0,0,3,3
1499 from_wm_2!TC,1,0,4,1,1,0
1501 from_wm_3!TC,1,0,4,1,1,0
1502 from_wm_local?TC,1,0,4,1,1,0
1513 from_wm_local?TC,1,0,4,1,1,0
1525 to_wm?TC,0,0,0,4,1,3,0
1531 from_wm_2!TC,0,0,4,1,3,0
1533 from_wm_3!TC,0,0,4,1,3,0
1534 from_wm_local?TC,0,0,4,1,3,0
1545 from_wm_local?TC,0,0,4,1,3,0
1559 to_wm!TC,1,1,2,3,2,1,2
1561 to_wm?TC,1,1,2,3,2,1,2
1567 from_wm_2!TC,1,2,3,2,1,2

Spin Version 6.1.0 -- 4 May 2011 -- solsr.pml -- MSC -- 6


0::init:
Wireless_Medium
Node
3:Node
4:Node
5:ByzNode
1569 from_wm_3!TC,1,2,3,2,1,2
1570 from_wm_local?TC,1,2,3,2,1,2
1583 to_wm!TC,2,2,1,12,2,0,1
1585 to_wm?TC,2,2,1,12,2,0,1
1587 from_wm_0!TC,2,1,12,2,0,1
1589 from_wm_1!TC,2,1,12,2,0,1
1594 from_wm_local?TC,1,2,3,2,1,2
1596 from_wm_local?TC,2,1,12,2,0,1
1609 to_wm!TC,2,2,0,4,1,0,0
1611 to_wm?TC,2,2,0,4,1,0,0
1613 from_wm_0!TC,2,0,4,1,0,0
1615 from_wm_local?TC,2,1,12,2,0,1
1636 from_wm_1!TC,2,0,4,1,0,0
1641 from_wm_local?TC,2,0,4,1,0,0
1643 from_wm_local?TC,2,0,4,1,0,0
1656 to_wm!TC,0,0,1,12,2,0,1
1668 to_wm?TC,0,0,1,12,2,0,1
1674 from_wm_2!TC,0,1,12,2,0,1
1676 from_wm_3!TC,0,1,12,2,0,1
1677 from_wm_local?TC,0,1,12,2,0,1
1690 to_wm!TC,1,1,0,4,1,0,0
1691 to_wm?TC,1,1,0,4,1,0,0
1697 from_wm_local?TC,0,1,12,2,0,1
1699 from_wm_2!TC,1,0,4,1,0,0
1701 from_wm_3!TC,1,0,4,1,0,0
1702 from_wm_local?TC,1,0,4,1,0,0
1723 from_wm_local?TC,1,0,4,1,0,0
1733 timeout

Appendix B: Other Secure Routing Protocols

Three secure protocols are described in this appendix. The protocols are Secure AODV (SAODV), Secure Efficient Distance Vector (SEAD), and Secure Link State Routing Protocol (SLSP). These protocols were not modeled with the secure verification framework. A logical progression would be to model SLSP next because it shares several common features with OLSR and Secure OLSR. SEAD is similar to Secure OLSR because it is proactive and SAODV is related because it is table-based. The security-mechanisms employed in each protocol are significantly different.

B.1 Secure AODV (SAODV)

Secure AODV (SAODV) [23] is based closely on AODV. The protocol's objective is to meet import authorization, source authentication, integrity, and data authentication for data used in the routing process. SAODV has several admitted weaknesses. The protocol cannot account for Byzantine nodes, cannot provide non-repudiation, and cannot prevent a malicious node from tunneling or the invisible node attack (INA) (i.e., colluding attackers).

For the proper operation of SAODV, it is assumed that a key management system exists, for example PKI. In addition, a node's identity is verifiable through that node's public key. Finally, to simplify the analysis, it is assumed that intermediate nodes cannot provide previously known paths to RReqs. (SAODV's Double Signature Extension can allow intermediate nodes to securely cache routes to RReqs.)

At a high-level SAODV meets its objectives by digitally signing all immutable fields. All fields in each RREQ and RREP are considered immutable aside from the hop count. For the one mutable field in AODV, hop count, one-way hash chaining can be used to assure that the field never decreases as it is travels toward the destination in a route request.

Specifically when SAODV generates a RReq or a RRep the fields Hash Function, Max Hop Count, Top Hash, and Hash are initialized. Max Hop Count is initially set to the time-to-live, a specific Hash Function is set, the Hash is set to a random integer, and the Top Hash is evaluated by performing the Hash Function Max Hop Count times on the initial Hash value.

When a node receives a RReq or RRep the hash of the hop count is computed and forwarded with all other information. SAODV requires that every node first verify the signature of the RReq or RRep messages before being processed. If the verification is successful then AODV processed the messages as normal plus the signing and hashing functions as described in [23].

The modifications to AODV proposed in SAODV do not significantly change the underlying operation of the protocol. The only new fields defined by SAODV are Hash Function, Max Hop Count, Top Hash, and Hash. The adding of these fields would add a constant overhead in state-vector size to the Promela models; therefore, it remains feasible to model SAODV in Promela and perform SPIN verifications on such a model.

As a final note, the discussion of SAODV omits RErr messages. The Promela model makes this same omission. It is reasonable in the models to assume that mobility is not a consideration. Link breakages are the only reason for triggering a RErr message, thus all models only consider a static topology for the network. Similar logic is used in Ács [19] to provide a proof for the endairA protocol. [7]

B.2 Secure Efficient Distance Vector (SEAD)

Secure Efficient Distance Vector (SEAD) [21] is based on DSDV, a proactive protocol which uses a destination-sequence number to prevent routing loops. SEAD is not considered to be secure against more than two malicious attackers that collude. The

[7] The assumption does not imply there are no issues with mobility. Quite the opposite, if it can be shown that the protocol is insecure without mobility then the protocol remains insecure even after adding mobility to the model.

primary goal of SEAD is to protect the destination-sequence number and minimum route length thus protecting the loop freedom property of the protocol.

Let m represent the shortest possible route length and s represent the current sequence number. SEAD using hash trees assures that an attacker can advertise at best a shortest path of $m + 1$. To achieve this goal SEAD utilizes one-way hash chains and Merkle hash trees. One-way hash chains guarantee that the lower bound of path length, m, cannot be lessened by Byzantine node. This guarantee, however, allows an adversary to advertise another route with distance m, Merkle hash trees can be used to further ensure an adversary cannot advertise a path less than $m + 1$.

The one way hash-chain is calculated by the recurrence relation $h_i = H(h_{i-1})$ where $h_0 = x$ and x is a random integer and i is the current node. The hash function ensures that nodes down stream of i cannot modify the chain without completely corrupting it. A corrupt hash chain can easily be identified and the associated packets discarded.

Merkle hash trees are calculated recursively from a parent root, m_p, according to $m_p = H[m_l \| m_r]$. The expression, $m_p = H[m_l \| m_r]$, reads the parent value m_p is equal to the one-way hash of the concatenation of m_l and m_r. Given this construct it is possible for every pair of nodes to calculate a hash based solely on information provided by the previous node. This property ensures that an adversary cannot simply rebroadcast the last hash value in the chain and pass it off as being valid because the recipient can verify that the adversarial node is not who he claims to be.

DSDV is not examined from a modeling perspective; however, SEAD does not add much additional burden to the protocol. For implementing Merkle trees additional hashing information must be passed which is, in the worst case, linear to the networks diameter. Thus if modeling DSDV is feasible using formal methods then so to is modeling SEAD.

Table B.1: link-state update (LSU) fields.

Field	**Purpose**
R_{LSU}	Track current hops this update has traversed
Zone Radius	X_R component of Hash chain
Sequence	Uniquely identify the LSU paired with origin address
Signature	Public key of the LSU's origin
Hops Traversed	X_1 component of Hash chain

B.3 Secure Link State Routing Protocol (SLSP)

Papadimitratos [24] proposes the Secure Link State Routing Protocol (SLSP). The protocol can be adapted to meet the security challenges of hybrid routing protocols incorporating both reactive and proactive sub-protocols.

SLSP's security considerations are limited to individual Byzantine attackers. The protocol designers do not claim that the protocol can be considered secure when challenged by two or more malicious nodes that collude. The protocol is considered to prevent advertisement of non-existent or fabricated link-states, prevent spoofing of peers, thwart Denial of Service (DoS) flooding attacks, and strengthen the over all neighbor discovery process.

The basic components of SLSP are link-state updates (LSUs) (equivalent to TC messages), neighbor look-up protocol (NLP) notifications, the hash chaining function $H()$, and key certification. Key certification is assumed to be provided by threshold cryptography, local repositories, or a distributed certificate authority (CA). $H()$ is defined as $X_i = H^i(X), i = 1, ..., R$, where R is the network diameter and $H^0(X) = X$. An LSU consists of the fields in Table B.1. SLSP modifies each LSU field as the LSU propagates through the network.

In SLSP nodes are bound to a single, unique identifying address and public key. Neighboring nodes can be unambiguously identified based on their public key, i.e., mapped to their unique address.

SLSP operates in the following fashion. SLSP performs neighbor discovery by exchanging signed HELLO messages between on-hop nodes. Only validated HELLO messages are accepted. The NLP then runs and specifically maintains a mapping of one-hop neighbors, identifies discrepancies, and measures the control packet rates. NLP will generate a notification if a neighbor uses an address different from the one recorded in the neighbor set or two neighbors use the address. Any packets from a source that triggered a notification are discarded. Nodes receiving an LSU validate the update, suppress duplicate updates, and relay previously un-relayed LSUs. A validated LSU update is only committed if the both nodes advertise the same state of the link (i.e., the link is bi-directional).

SLSP provides a packet format for public key distribution. However, LSUs are also able to perform key distribution; therefore, discussion of the public key distribution packet is unnecessary.

Notably SLSP differs from OLSR in several respects. SLSP provides no concept of MPRs and in fact uses this as a security feature. By implementing pure flooding SLSP ensures that a non-adversarial route, if once exists, will be found. In the same vein SLSP has no requirement to discovery two-hop neighbors, thus making SLSP considerably simpler to model than OLSR.

Appendix C: Acronyms

ANSN advertised neighbor sequence number 15

AODV Ad hoc On-demand Distance Vector 12

ARAN Authenticated Routing for Ad Hoc Networks 23

AVANTSSAR Automated Validation of Trust and Security of Service-oriented Architectures 30

AVISPA Automated Validation of Internet Security Protocols and Applications 27

CA certificate authority 89

CIA confidentiality, integrity, and availability 19

CL-AtSe Constraint Logic-Attack Searcher 30

CPAL-ES Cryptographic Protocol Analysis Language - Evaluation System 27

CSMA/CA Carrier Sense Multiple Access/Collision Avoidance 39

DSDV Destination Sequence Distance Vector 12

DSR Dynamic source routing 12

DYMO Dynamic MANET On-demand Routing 6

DoS Denial of Service 89

GloMoSim Global Mobile Simulator 26

IEEE Institute of Electrical and Electronics Engineers 39

IETF Internet Engineering Task Force 6

INA invisible node attack 86

LSU link-state update 89

LTL Linear temporal logic 32

LUNAR Lightweight Underlay Network for Ad hoc Routing 12

MANET mobile ad hoc network 5

MPR multi-point relay 13

NLP neighbor look-up protocol 89

NS2 Network Simulator 2 26

OFMC On-the-Fly Model Cheker 30

OLSR Optimized Link-State Routing 5

OSI model Open Systems Interconnection model 21

PKI Public Key Infrastructure 19

Promela Process Metalanguage 29

RREP route reply 13

RREQ route request 13

SAODV Secure AODV 23

SATMC SAT-based Model-Checker 30

SEAD Secure Efficient Distance Vector 23

SLSP Secure Link State Routing Protocol 23

SMF Simplified Multicast Forwarding 6

SPIN Simple Promela Interpreter 5

SRP Secure Routing Protocol 29

TC topology control 15

TTL time to live 16

VANET vehicular ad hoc network

WG Working Group 6

WMN wireless mesh network 9

WSN wireless sensor network

Bibliography

[1] D.M. Ritchie and B.W. Kernighan. *The C programming language*. Bell Laboratories, 1988.

[2] Todd R. Andel. *Formal Security Evaluation of Ad Hoc Routing Protocols*. PhD thesis, Florida State University, 2007.

[3] Norman Abramson. The ALOHA system–another alternative for computer communications. In *Fall Joint Computer Conference*, 1970.

[4] G. Ludwig and R. Roy. Saturation routing network limits. *Proceedings of the IEEE*, 65(9):1353 – 1362, sept. 1977.

[5] Eli M. Gafni and Dimitri P. Bertsekas. Distributed Algorithms for Generating Loop-Free Routes in Networks with Frequently Changing Topology. In *IEEE Transactions on Communications*, volume 29 of *IEEE Transactions on Communications*, pages 11–18. IEEE, January 1981.

[6] SECAN LAB. Ad Hoc Routing Protocols (History), 2011.

[7] P. Gupta and P.R. Kumar. The capacity of wireless networks. *Information Theory, IEEE Transactions on*, 46(2):388 –404, March 2000.

[8] F. Stajano and R. Anderson. The resurrecting duckling: Security issues for ad-hoc wireless networks. In *Security Protocols*, pages 172–182. Springer, 2000.

[9] Chris Karlof and David Wagner. Secure routing in wireless sensor networks: attacks and countermeasures. *Elsevier: Ad Hoc Networks*, 1:293–315, 2003.

[10] Todd R. Andel and Alec Yasinsac. Surveying security analysis techniques in MANET routing protocols. *IEEE Communications Surveys and Tutorials*, 9:70–84, 2007.

[11] Zygmunt J. Haas, Marc R. Pearlman, and Prince Samar. The Zone Routing Protocol (ZRP) for Ad Hoc Networks. Internet-draft, IETF MANET Working Group, July 2002. Expiration: January, 2003.

[12] D.B. Johnson and D.A. Maltz. Dynamic source routing in ad hoc wireless networks. *Mobile computing*, pages 153–181, 1996.

[13] I. Chakeres and C. Perkins. Dynamic manet on-demand (dymo) routing. *draft-ietf-manet-dymo-19. txt (work in progress)*, 2010.

[14] C. Perkins, E. Belding-Royer, and S. Das. Ad hoc On-Demand Distance Vector (AODV) Routing. RFC 3561 (Experimental), July 2003.

[15] O. Wibling, J.Parrow, and A. Pears. Automatized verification of ad-hoc routing protocols, in: Formal Techniques for Networked and Distributed Systems. *FORTE 2004*, 3235:343–358, 2004.

[16] T. Clausen and P. Jacquet. Optimized Link State Routing Protocol (OLSR). RFC 3626, IETF, October 2003.

[17] Charles E. Perkins and Pravin Bhagwat. Highly dynamic Destination-Sequenced Distance-Vector routing (DSDV) for mobile computers. In *Proceedings of the conference on Communications architectures, protocols and applications*, SIGCOMM '94, pages 234–244, New York, NY, USA, 1994. ACM.

[18] Amir Qayyum, Laurent Viennot, and Anis Laouiti. Multipoint relaying for flooding broadcast messages in mobile wireless networks. In *Proceedings of the 35th Hawaii International Conference on System Sciences - 2002*, page 298, 2002.

[19] Gergely Acs, Levente Buttyan, and Istvan Vajda. Provably Secure On-Demand Source Routing in Mobile Ad Hoc Networks. *IEEE Transactions on Mobile Computing*, 5:1533–1546, November 2006.

[20] Y.C. Hu, A. Perrig, and D.B. Johnson. Ariadne: A secure on-demand routing protocol for ad hoc networks. *Wireless Networks*, 11(1):21–38, 2005.

[21] Yih-Chun Hu, David B. Johnson, and Adrian Perrig. SEAD: secure efficient distance vector routing for mobile wireless ad hoc networks. *Ad Hoc Networks*, 1(1):175 – 192, 2003.

[22] K. Sanzgiri, B. Dahill, B.N. Levine, C. Shields, and E.M. Belding-Royer. A secure routing protocol for ad hoc networks. In *Network Protocols, 2002. Proceedings. 10th IEEE International Conference on*, pages 78 – 87, November 2002.

[23] M.G. Zapata and N. Asokan. Securing ad hoc routing protocols. In *Proceedings of the 1st ACM workshop on Wireless security*, pages 1–10. ACM, 2002.

[24] P. Papadimitratos and Z. J. Haas. Secure link state routing for mobile ad hoc networks. *IEEE Workshop on Security and Assurance in Ad hoc Networks and 2003 International Symposium on Applications and the Internet, Orlando, FL, January 28, 2003*, 2003.

[25] C. Adjih, T. Clausen, P. Jacquet, A. Laouiti, P. Muhlethaler, and D. Raffo. Securing the OLSR protocol. In *Proceedings of Med-Hoc-Net*, pages 25–27. Citeseer, 2003.

[26] Davide Benetti, Massimo Merro, and Luca Vigano. Model Checking Ad Hoc Network Routing Protocols: ARAN vs. endairA. Technical report, Dipartmento di Informatica, Universita degli Studi di Verona, Italy, 2010.

[27] A. Singh, CR Ramakrishnan, and S.A. Smolka. A process calculus for mobile ad hoc networks. *Science of Computer Programming*, 75(6):440–469, 2010.

[28] L. Buttyán and Ta Vinh Thong. Formal verification of secure ad-hoc network routing protocols using deductive model-checking. In *Wireless and Mobile Networking Conference (WMNC), 2010 Third Joint IFIP*, pages 1 –6, oct. 2010.

[29] D. Dolev and A. Yao. On the security of public key protocols. *Information Theory, IEEE Transactions on*, 29(2):198 – 208, March 1983.

[30] Alan M. Turing. On computable numbers, with an application to the entscheidungsproblem. *Proceedings of the London Mathematical Society*, 42:230–265, 1936.

[31] Gerard J. Holzmann. *The SPIN Model Checker*. Addison Wesley, 2004.

[32] Mihai-Lica Pura, Victor-Valeriu Patriciu, and Ion Bica. Formal verification of secure ad hoc routing protocols using AVISPA: ARAN case study. *Proceedings of the 4th European Computing Conference*, 4:200–206, 2010.

[33] Theo C. Ruys. *Towards Effective Model Checking*. PhD thesis, Twente University, 2001.

[34] T.R. Andel, G. Back, and A. Yasinsac. Automating the security analysis process of secure ad hoc routing protocols. *Simulation Modelling Practice and Theory*, 19(9):2032–2049, 2011.

[35] Todd R. Andel and Alec Yasinsac. The Invisible Node Attack Revisited. *IEEE Computer*, 2007:686–691, 2007.

[36] C. Adjih, T. Clausen, A. Laouiti, P. Muhlethaler, and D. Raffo. Securing the OLSR routing protocol with or without compromised nodes in the network. *INRIA RR-5747, Nov*, 2005.

REPORT DOCUMENTATION PAGE	*Form Approved* *OMB No. 0704–0188*

1. REPORT DATE *(DD–MM–YYYY)* 22–03–2012	**2. REPORT TYPE** Master's Thesis	**3. DATES COVERED** *(From — To)* Aug 2010 — Mar 2012

4. TITLE AND SUBTITLE Security Verification of Secure MANET Routing Protocols	**5a. CONTRACT NUMBER** **5b. GRANT NUMBER** F1ATA01103J001 **5c. PROGRAM ELEMENT NUMBER**
6. AUTHOR(S) Steele, Matthew F., Captain, USAF	**5d. PROJECT NUMBER** 12G292V **5e. TASK NUMBER** **5f. WORK UNIT NUMBER**
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Air Force Institute of Technology Graduate School of Engineering and Management (AFIT/EN) 2950 Hobson Way WPAFB OH 45433-7765	**8. PERFORMING ORGANIZATION REPORT NUMBER** AFIT/GCS/ENG/12-03
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) AFOSR/RSL Dr. Robert L. Herklotz 875 N. Randolph Street, Suite 325, Room 3112 Arlington, VA 22203-1768 (703) 696-6565; robert.herklotz@afosr.af.mil	**10. SPONSOR/MONITOR'S ACRONYM(S)** Information Operations and Security AFOSR (AFOSR/RSL) **11. SPONSOR/MONITOR'S REPORT NUMBER(S)**

12. DISTRIBUTION / AVAILABILITY STATEMENT

Distribution A. Approved for Public Release; Distribution Unlimited

13. SUPPLEMENTARY NOTES

This material is declared a work of the United States Government and is not subject to copyright protection in the United States.

14. ABSTRACT

Secure mobile ad hoc network (MANET) routing protocols are not tested thoroughly against their security properties. Previous research focuses on verifying secure, reactive, accumulation-based routing protocols. An improved methodology and framework for secure MANET routing protocol verification is proposed which includes table-based and proactive protocols. The model checker, SPIN, is selected as the core of the secure MANET verification framework. Security is defined by both accuracy and availability: *a protocol forms accurate routes and these routes are always accurate*. The framework enables exhaustive verification of protocols and results in a counter-example if the protocol is deemed insecure. The framework is applied to models of the Optimized Link-State Routing (OLSR) and Secure OLSR protocol against five attack vectors. These vectors are based on known attacks against each protocol. Vulnerabilities consistent with published findings are automatically revealed. No unknown attacks were found; however, future attack vectors may lead to new attacks. The new framework for verifying secure MANET protocols extends verification capabilities to table-based and proactive protocols.

15. SUBJECT TERMS

formal methods, Secure OLSR, verification, security, mobile ad hoc network

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON**
a. REPORT	**b. ABSTRACT**	**c. THIS PAGE**			Maj Todd R. Andel
					19b. TELEPHONE NUMBER *(include area code)*
U	U	U	UU	108	(937) 255–3636, ext 4901; todd.andel afit.edu